Tampere University

SERGIO MORESCHINI

Applications of MLOps in the Cognitive Cloud Continuum

Tampere University Dissertations 861

Tampere University Dissertations 861

SERGIO MORESCHINI

Applications of MLOps in the Cognitive Cloud Continuum

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Technology and Communication Sciences
of Tampere University,
for public discussion in the Auditorium room TB109
of the Tietotalo, Korkeakoulunkatu 1, Tampere,
on 17 November 2023, at 12 o’clock.

ACADEMIC DISSERTATION
Tampere University, Faculty of Information Technology and Communication Sciences
Finland

ISBN 978-952-03-3053-8 (print)
ISBN 978-952-03-3054-5 (pdf)
ISSN 2489-9860 (print)
ISSN 2490-0028 (pdf)
http://urn.fi/URN:ISBN:978-952-03-3054-5

HIILINEUTRAALI PAINOTUOTE
ClimateCalc CC-000025/FI
PunaMusta Printing

Carbon dioxide emissions from printing Tampere University dissertations have been compensated.

PunaMusta Oy – Yliopistopaino
Joensuu 2023

Non est ad astra mollis e terris via

Lucio Anneo Seneca, *Hercules furens*

ACKNOWLEDGEMENTS

This thesis is the second and biggest chapter of a journey that began in January 2014. The presented publications were done within the CloudSEA group.

The first person I want to thank is Associate Professor David Hästbacka for the meaningful and stimulating discussions and for creating a healthy and conducive working environment.

My deepest gratitude goes to my co-supervisor, Professor Davide Taibi, for being a leader, a mentor and, outside the working environment, a friend. At the same level, I would like to thank my *unofficial* supervisor, Assistant Professor Valentina Lenarduzzi, who had the courage to bet on me when not even I could. Davide and Valentina gave me the opportunity to appreciate different aspects of academic life, they created a strong team that allowed me to be productive at my best and showed me the importance of research for both technological progress and universal knowledge. Grazie, or as they would appreciate more: *"Daje!"*

I would also like to thank all the people who have been directly and indirectly involved in my academic life over the last 7 years, starting with my team CloudSEA for the amazing ride together, especially Xiaozhou and Nyyti, and the former Department of Signal Processing at Tampere University of Technology for the life lessons.

On a more personal side the first person I would like to thank is Francesco for holding me up and being always ready for discussions, feedbacks and challenges being both my closest colleague and a true friend. Together with him my gratitude goes to Barbara, Michelangelo, Chiara, Donato, and Federica who have been my Finnish family for the last 6 years.

Following I would like to thank my Spanish family and my teams Vierasjoukkue, i Pirlalla, la Vecchia Guardia, and Gelarmn, in particular Alberto, Jose, Laura, Irene, Ruben, Zio, Davide, Marco, Lucio, Maggiu, Mazzi, Claudio, Marianna, Bota and Gino.

A special mention goes to those who never made me feel the distance as an insurmountable limit: Andrea, Federica, Marta V., Daniele, Marta D.F., Lisa, Claudia, Audrey, Davide.

To those who have always been close in my *After Hours*: Palmone, Giusy, Ashley, and Erika.

I would like to thank Miriana, without whom I would never have understood the importance of going forward and thus this work would not exist.

Last but not least my *Bro* Andrea who shared a journey parallel to mine. We were fortunate enough to share some of the most important adventures in these 9 years and half, from concerts to flats. **We Are Not Your Kind**.

A particular and meaningful mention goes to my parents and relatives who were able to accept every choice I made in my life and supported me unconditionally, especially during long and hard times. I know you will be by my side also for the next chapters.

Tampere, August 2023
Sergio Moreschini

ABSTRACT

In the last decade, the development of software based on artificial intelligence has increased exponentially. The adoption of techniques based on the use of machine learning and deep learning in particular, has made it possible to develop applications and create previously unthinkable solutions. The evolution led by the use of machine learning over classical software development has required new guidelines for the software development lifecycle. While DevOps (a blend of the words "DEVelopment" and "it OPerationS") has traditionally been the standard guideline for software development, it lacks certain steps required by machine learning. This led to the rise of its natural evolution: MLOps (a combination of the words "Machine Learning" and "it OPerationS").

This thesis investigates the application of MLOps in the Cognitive Cloud Continuum by exploiting both empirical methodologies and practical applications in the field of machine learning for software engineering. To do so the primary objective is to identify the risk of embedding open source libraries when developing machine learning-based applications, in particular when following the MLOps principles. Secondly, we aim at investigating what is the Cognitive Cloud Continuum and what are the future implications. Finally, by showing different use cases we compare the differences between the development of software not based on machine learning and the development of software following the MLOps guidelines.

The results show that it is possible to compute the risk of embedding open-source software (and therefore libraries) when developing machine learning-based code. The analysis of the literature has identified an increased interest in the concept of Cognitive Cloud Continuum, which has resulted in the development of tools aimed at increasing the level of automation in order to comply with the new requirements of this concept. The results achieved have been showing that the concepts provide promising results. The thesis has successfully demonstrated how to develop a model based on an MLOps pipeline.

This thesis contributes to the state of the research by increasing awareness of the concept of Cognitive Cloud Continuum and emphasizing the importance of following MLOps guidelines when developing ML-based software. In the future, devices will be capable of exploiting the computational power of other entities being part of the same environment to carry out multiple tasks such as continuous training and deployment. The required level of automation in the software development lifecycle will be based on the guidelines defined by MLOps.

CONTENTS

1 Introduction 1

1.1 Goal and Research Questions 2

1.2 Research Methods and Contributions 3

1.3 Thesis Structure 4

2 Background 5

2.1 Software Development Lifecycle 5

2.1.1 DevOps 6

2.1.2 Machine Learning 11

2.1.3 MLOps 13

2.2 Software Architectures 16

2.2.1 Computation Distribution 17

3 Research Methods 19

3.1 Publication I - Survey 19

3.2 Publication II - Model development 21

3.3 Publication III - Systematic Mapping Study 25

3.4 Publication IV - Algorithm Development 27

3.5 Publication V - MLOps Pipeline Creation 30

4 Results and Research Contribution 33

4.1 RQ1: What are the potential issues of embedding Open Source Libraries in MLOps?
Publication I - Publication II 33

4.2 RQ2: What is the Cognitive Cloud Continuum?
Publication III 36

4.2.1 Cognitive Cloud 39

4.2.2 Cognitive Cloud Continuum 40

4.3 RQ3: How does the software development process change when doing MLOPs?
PUBLICATION IV - PUBLICATION V. 40
4.3.1 Static Case. 41
4.3.2 Dynamic Case. 42

5 Discussion 45
5.1 Answers to research questions. 45
5.2 Discussion of the results. 47
5.3 Threats to Validity. 48

6 Conclusion 51

References 53

Appendix A Publication Summary. 67

Publication I 71

Publication II 93

Publication III 101

Publication IV 115

Publication V 129

List of Figures

1.1 Illustration of the contributions of the thesis. 2

2.1 The DevOps Lifecycle infinite loop 7
2.2 The differences between AI, ML, and DL in Data Science 12
2.3 Different branches derived from DevOps. Reprinted with permission from PUBLICATION V. 13
2.4 The MLOps Lifecycle infinite loop. Reprinted with permission from [62]. ©IEEE 2022. 15

3.1 The OSSARA process. Reprinted with permission from Publication II. 23
3.2 The data pipeline. Reprinted with permission from Publication V.. . 30

4.1 Definitions of cloud computing grouped by year and concepts. Each column represents a different year while the colored blocks represent different aspects. Arrows between two blocks indicate that there is a direct citation to the definition. Reprinted with permission from Publication III. 37
4.2 Example of a system with multiple *locations* (L1, L2, and L3), each with multiple *edge nodes* (E1, E2, ..), and a variety of *IoT devices* connected to the the edge nodes of the closest location (squares with same colors as locations). Moreover, the lines connecting edge nodes of different locations indicate the possibility to handover the connections of the IoT devices. Reprinted with permission from Publication IV. ©Springer Nature 2023. 41

List of Tables

4.1 High-level factors considered during the adoption of OSS. Reprinted with permission from Publication I. 34

5.1 Research Questions summary: RQs and the results achieved. 46

ABBREVIATIONS

AI	Artificial Intelligence
CD	Continuous Delivery
CI	Continuous Integration
COCLCON	Cognitive Cloud Continuum
CT	Continuous Training
DEVOP	Development & Operations
DL	Deep Learning
et al.	and others, from Latin *et alii*
FaaS	Function as a Service
IDE	Integrated Development Environments
IoT	Internet of Things
ML	Machine Learning
MLOPS	Machine Learning & Operations
MLR	Multivocal Literature Review
MS	Microservice
NIST	National Institute of Standards and Technology
OSS	Open Source Software
OSSARA	OSS Abandonment Risk Assessment
RAT	Radio Access Technology
SDLC	Software Development Life Cycle
SLR	Systematic Literature Review

SOA	Software-oriented Architecture
SP	Selected Papers

ORIGINAL PUBLICATIONS

Publication I	Xiaozhou Li, Sergio Moreschini, Zheying Zhang, and Davide Taibi. "Exploring factors and metrics to select open source software components for integration: An empirical study". In: *Journal of Systems and Software* 188 (2022), p. 111255. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2022.111255.
Publication II	Xiaozhou Li, Sergio Moreschini, Fabiano Pecorelli, and Davide Taibi. "OSSARA: Abandonment Risk Assessment for Embedded Open Source Components". In: *IEEE Software* 39.4 (2022), pp. 48–53. DOI: https://doi.org/10.1109/MS.2022.3163011.
Publication III	Sergio Moreschini, Fabiano Pecorelli, Xiaozhou Li, Sonia Naz, David Hästbacka, and Davide Taibi. "Cloud Continuum: The Definition". In: *IEEE Access* 10 (2022), pp. 131876–131886. DOI: https://doi.org/10.1109/ACCESS.2022.3229185.
Publication IV	Sergio Moreschini, Francesco Lomio, David Hästbacka, and Davide Taibi. "Smart Edge Service Update Scheduler: An Industrial Use Case". In: *Service-Oriented Computing – ICSOC 2022 Workshops*. Springer Nature Switzerland, 2023, pp. 171–182. DOI: https://doi.org/10.1007/978-3-031-26507-5_14.
Publication V	Sergio Moreschini, David Hästbacka, and Davide Taibi. "MLOps pipeline development: the OSSARA use case". In: *Proceedings of the Conference on Research in Adaptive and Convergent Systems*. 2023. In press.

Author's contribution

The contribution of the author, Sergio Moreschini, in the publication presented in this work, is described below.

Publication I	The author participated in all the phases of this work, including the planning, creation, conduction of the interviews, development of the software used to investigate the APIs of the portals, the formalization of the analysis performed as well as the investigation performed. Finally the authors led the writing process. The author equally shared the lead authorship with another author.
Publication II	The author participated in all the phases of this work, including the conceptualization and planning of the work. He took part in the development of the model presented in the work and the validation performed. The author led the writing process. The author equally shared the lead authorship with another author.
Publication III	The author participated in the whole planning, conceptualization, and development of the research methodology of the work. The author led the search process, selection, and data extraction of the primary studies. He also led the process of analysis and interpretation of the results achieved from the data extracted by the primary studies. The author participated in the overall writing process. The author is the lead author of the publication.
Publication IV	The author participated in the overall conceptualization and planning of the study. The author took charge of planning, researching, and executing suitable techniques for the data analysis. He also took care of the interpretation of the results. The author participated in the overall writing process. The author is the lead author of the publication.
Publication V	The author took charge of the planning and conceptualization of the work. The author took charge of planning, researching, and executing suitable techniques for the data analysis. He also took

care of the interpretation of the results. The author participated in the overall writing process. The author is the lead author of the publication.

1 INTRODUCTION

In the last decade, we have witnessed a revolution in the software development process. In the specific, the Software Development Life Cycle (SDLC) [27], originally composed of its four specific phases, has once again evolved. After the introduction of Agile, DevOps was introduced [7], but nowadays we need an additional concept. The necessity of such evolution came into play with the broad development of Machine Learning (ML) based applications. ML-based applications have substituted conventional applications due to their capability in solving challenges based on the use of data [56]. Therefore, as a new approach to creating applications required different steps, an evolution of DevOps (which stands for Software Developers and IT operations working collaboratively) adapted to ML has become necessary. Such an evolutionary approach has been defined as *MLOps* as it stands for Machine Learning and IT operations. This new approach has increased the accuracy of the results achieved by the software and decreased the time necessary for its creation [18].

In this thesis, we will refer to ML-based software/applications when describing the process of creating applications that do not need to be deployed but can also run locally. On the contrary, we will refer to MLOps as an extension of ML-based software that also includes the build and deployment of the software and all Ops phases.

The creation of ML-based software has been made possible by the maturation of new technologies and, in particular, the availability of new and more powerful hardware. The widespread availability of open-source, ready-to-integrate libraries to simplify code development has also been an important factor. This resulted in the creation of new applications for multiple fields e.g. self-driving cars or video frame enhancements (also known as frame reconstruction) in Computer Vision [73] or fault detection in Software Engineering [51].

The newly developed hardware opened multiple possibilities not only for what concerns the computational power of the devices but also its portability. The dif-

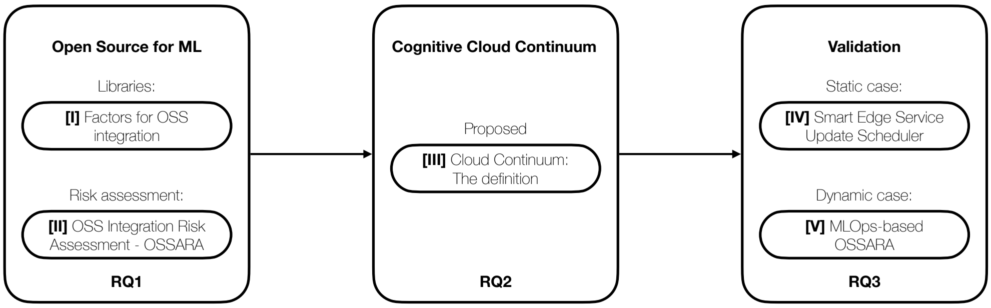

Open Source for ML
Libraries:
[I] Factors for OSS integration
Risk assessment:
[II] OSS Integration Risk Assessment - OSSARA
RQ1
Cognitive Cloud Continuum
Proposed
[III] Cloud Continuum: The definition
RQ2
Validation
Static case:
[IV] Smart Edge Service Update Scheduler
Dynamic case:
[V] MLOps-based OSSARA
RQ3

Figure 1.1 Illustration of the contributions of the thesis.

ferent boundaries that were once creating prominent clusters of devices based on the tradeoff between computational power and portability have nowadays been broken. The result is a new concept defined as the *Cognitive Cloud Continuum* (COCLCON) based on the definitions of Cognitive Cloud [64] and Cloud Continuum [65].

The aim of this thesis is to provide insight into the development of ML-based software and the risk of embedding open source libraries in its development. In particular, the focus is on providing examples of how to develop software following the MLOps guidelines. We are also interested in software that can be deployed along the COCLCON, ensuring both portability and computational power. To this end, a clear definition of the COCLCON is one of the main objectives of this work.

1.1 Goal and Research Questions

The goal of this thesis is bivalent. On one side our goal is to guide users in the process of **creating MLOps-based software**. Once detected the main phases composing MLOps, we aim at exploiting tools to simplify the software development for each step of the MLOps pipeline.

In parallel, we aim to **define the concept of the Cognitive Cloud Continuum**. After identifying the main characteristics of the COCLCON we aim at exploiting them to provide guidelines to develop ML-based software along the COCLCON.

To achieve such goals we answer multiple research questions (RQ_s):

RQ_1 - What are the potential issues of embedding Open Source Libraries in MLOps?

The first research question aims at assessing the risk of embedding libraries when developing ML-based code. By treating ML libraries as Open Source Software (OSS), first, the factors and metrics used for OSS selection will be studied. Following, a model for predicting the risk of abandonment of OSS will be presented.

RQ$_2$ - What is the Cognitive Cloud Continuum?

The second research question aims at providing insights into the development of software in the COCLCON. To properly study the process of development along the whole COCLCON it is important to understand first what are the main requirements to develop software in the COCLCON. For this reason a clear and unified definition of the COCLCON will be presented.

RQ$_3$ - How does the software development process change when doing MLOps?

As a last research question, we will develop a use case not based on ML and therefore defined as "static". Subsequently, such a static use case will be compared with a "dynamic" use case. The dynamic use case will be developed according to the MLOps guidelines.

1.2 Research Methods and Contributions

The research conducted in this thesis includes both empirical methodology and use cases. The empirical methodology includes systematic literature reviews, case studies, surveys, and interviews. The use cases are developed to validate the studies presented in the empirical methodology. More specifically, this thesis contributes to increasing knowledge about the risk of embedding open source libraries when developing ML-based applications. In addition, we pay special attention to the development of ML-based software following the MLOps guidelines as well as the definition of the COCLCON and its requirements.

In this thesis, after analyzing what are the most common factors and metrics used to select OSS, we propose a model to calculate the risk of embedding OSS. We then propose our vision on the concept of the Cognitive Cloud Continuum. Finally, using two different use cases, we show the differences in developing software based on AI and MLOps pipelines. The pipeline consists of tools aimed at simplifying the software development process that can be used to maximize the environment.

1.3 Thesis Structure

The remainder of the thesis is structured as follows. In Chapter 2, we present the background of the concepts presented in the thesis. Starting from the concept of the Software Development Lifecycle, we present its evolution focusing on DevOps. Later, to ease the transition from DevOps to MLOps, we present the basic concepts of ML. Following the main focus is placed on MLOps. From the proposed pipelines, the tools and their mapping to the different MLOps phases are presented. The second part of Chapter 2 focuses on Software Architecture and how to perform Computation Distribution, concepts that are useful in the definition of the Cognitive Cloud Continuum.

Chapter 3 describes the different research methods used in the different publications.

Chapter 4, introduces the results and research contribution of the work. The chapter is divided based on the different research questions that this work aims at answering.

Chapter 5 discusses the previously presented results.

Chapter 6 concludes the work and presents future challenges and works.

The peer-reviewed works are attached at the end of the thesis to help the reader in connecting the works to the research questions of the work.

2 BACKGROUND

In this chapter, we investigate the background of the concepts analyzed in this thesis. This is supported by articles in the literature. In Section 2.1.1, we describe the evolution of the SDLF, focusing specifically on DevOps. To ease the transition to the MLOps concept, in Section 2.1.2 we review the basics of ML and how nowadays the creation of ML-based applications is based on the use of Open Source Libraries.

2.1 Software Development Lifecycle

The Software Development Lifecycle (SDLC) is a methodology employed by software development teams to create software in a systematic and cost-efficient manner, ensuring high quality. The SDLC approach is implemented by software organizations of all sizes, utilizing various development models, such as agile, lean, waterfall, and others. Since 1960, the SDLC enables organizations to create effective software starting from the collection of initial product requirements by providing a structured and sequential method.

Initially, the model was devised by major corporations to handle intricate business systems that necessitated extensive data analysis and processing. With the passage of time, modified versions of this framework have been embraced for creating hardware and software technology products and other multifaceted projects [27]. Nowadays a very common practice when developing software is to compose software via the adoption of components off the shelf (COTS) [15]. Due to the integration of various embedded components, COTS can be considered as OSS, where by OSS we refer to software that is not commercial or proprietary in license. [34]. In particular these days, OSS is widely utilized and integrated into many commercial products. Nonetheless, selecting OSS projects for integration is not an easy task, primarily due to the absence of distinct selection models and insufficient information available on the OSS portals. Researchers have been investigating the selection and evaluation of

OSS from various angles. They have devised methods to assess, compare, and choose OSS projects, utilizing various approaches such as manual data extraction from OSS portals, as well as a range of tools and techniques.

However, software Development (Dev) is only part of the process of delivering a complete product. Security, deployment, monitoring, and feedback are only some of the processes that are not taken into account by the Dev team. Such responsibilities are performed by the Operations team (Ops).

Based on the ISO/IEC standards, we can divide the activities of software development from those of software operations:

- **Software Development:** software requirements analysis, software architectural design, software detailed design, software construction, software integration, and software qualification testing
- **Software Operations:** release and activation of the software product for operational use and establishing procedures for testing the software product in the operational environment, recording and resolving problems and modification requests of software product

2.1.1 DevOps

The importance of collaboration between the Dev and Ops entities is emphasized by the term DevOps. Different definitions of the DevOps concept have been presented and, even if no *unicum* exists, all of these definitions agree on the importance of such collaboration [7, 26, 78, 88, 33, 38]. The definition of DevOps used in this thesis is:

"DevOps is a set of practices that combines software development (Dev) and information technology operations (Ops) to enable organizations to deliver software products and services more quickly, reliably, and efficiently".

DevOps emphasizes collaboration and communication between development and operations teams, as well as the use of automation tools and agile methodologies to streamline the development and deployment process. The goal of DevOps is to create a culture of continuous integration, continuous delivery, and continuous improvement so that software development and deployment can be done rapidly, reliably, and with high quality [44, 37, 104, 52].

The continuous nature of DevOps is represented by the infinity loop employed

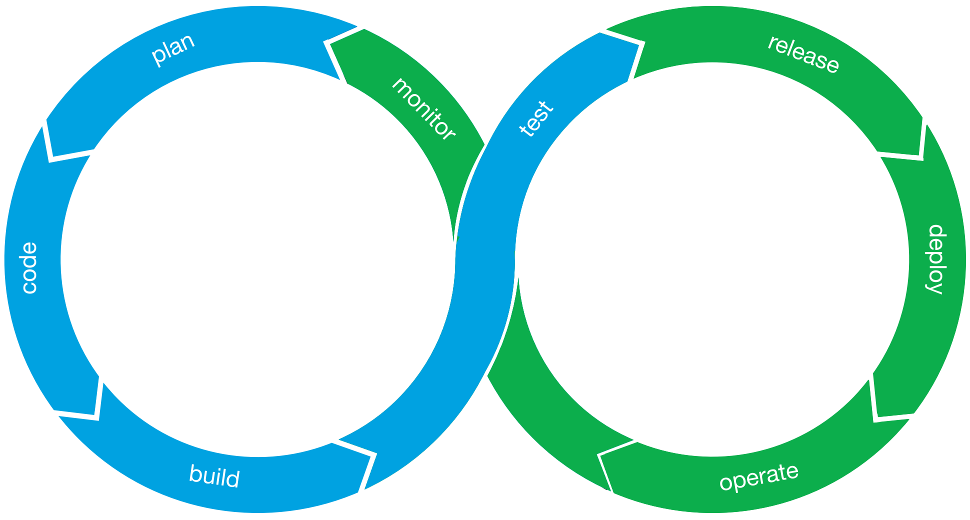


Figure 2.1 The DevOps Lifecycle infinite loop

by its practitioners (Figure 2.1). This loop illustrates the interrelatedness of the various phases in the DevOps lifecycle. Although the phases may seem to follow a linear sequence, the loop signifies the essential requirement for continuous collaboration and iterative enhancement throughout the entire lifecycle. The infinite loop is composed of 8 different phases, half of them representing Dev and half of them representing the Ops [23]. The phases are:

- **Dev:**
  - **Plan:** Before the Dev team commences writing code, the Plan stage encompasses all the activities that occur, and it is at this stage where the Product Manager or Project Manager demonstrates their value. The planning phase aims at creating a roadmap that outlines the project goals and deciding on the software and tools to be used are essential aspects of planning. This includes planning for the project's technology, environment, structure, and architecture.
  - **Code:** Once the planning has been completed the Dev team can start coding. This is a great opportunity to leverage automation tools as we prepare to build a testable product. Although coding can be time-consuming, automating certain tasks can help us optimize our time and resources. Developers use a variety of tools and techniques to write code, includ-

  ing version control systems, code editors, and integrated development environments (IDEs). During this phase, developers also conduct code reviews and collaborate with other team members to ensure that the code is high-quality and meets the project's requirements.
  - **Build:** Upon completion of a task, a developer shares their code by committing it to a shared code repository. There are various methods of doing this, but typically, the developer initiates a pull request - a request to integrate their fresh code with the shared codebase. During the Build phase, we compile the provided code in a development environment specifically for testing purposes. This allows us to identify and fix any issues or bugs in the code, ensuring that the software is stable and reliable.
  - **Test:** After a successful build, it is automatically deployed to a staging environment for thorough out-of-band testing. Automated testing is implemented to verify that the project is operating as intended and to identify and report any bugs or issues with its behavior. Such testing includes unit testing, integration testing, and system testing. Test automation tools are often used to speed up the testing process and improve the accuracy of test results.

- **Ops:**

  - **Release:** After undergoing a set of manual and automated tests, each code modification can be deemed reliable, and the operations team can have confidence that there is minimal possibility of breaking issues or regressions. At this stage, we have the first milestone of the project: the Release Phase. Organizations may opt to automatically deploy any build that reaches this stage of the pipeline, depending on their DevOps maturity level. On the other hand, an organization may prefer to exercise authority over the timing of releasing builds to production. They may wish to adhere to a fixed release schedule or launch new features only after achieving a significant milestone. To achieve this, a manual approval process can be implemented during the release stage, which only permits specific individuals within the organization to authorize a production release.
  - **Deploy:** The deployment phase involves releasing the software to pro-

duction. This phase involves a series of steps to deploy the software, including configuring servers, installing software dependencies, and deploying the code. Continuous delivery and deployment (CD/CD) tools are often used to automate the deployment process and reduce the risk of errors. Different deployment techniques can be used, among these, there are rolling deployment, blue-green deployment, canary deployment, and A/B Testing [86]. In the first, all nodes are incrementally updated in N batches. In the second, two identical environments are created, and once all tests are complete, traffic is moved from the working environment (green) to the test environment (blue). In the canary deployment, the new services are incrementally released to a subset of users, while in the A/B testing different versions of the same service run simultaneously as experiments in the same environment.

- **Operate:** At this state the freshly deployed release is live and available to the customers. The operations team is currently engaged in ensuring that everything runs seamlessly. Depending on the hosting service's configuration, the environment can automatically adjust to accommodate fluctuations in the number of active users, by scaling up or down as necessary. An essential part of the release phase is the feedback loop as the customer, being the world's best testing team, has a better understanding of their own needs and desires than anyone else.
- **Monitor:** The last phase of the pipeline leverages the feedback obtained from customers in the Operate phase, by gathering data and generating analytics on customer behavior, performance, errors, and other metrics. The gathered information is then shared with the whole team to complete the cycle of the process. This causes a new planning phase and, therefore, the loop to start again.

Following such guidelines has been proven to provide clear benefits to practitioners. Some of these benefits are [94]:

- **Faster time to market:** DevOps allows organizations to release software updates more frequently and with greater speed, which can lead to faster time to market and improved competitiveness.
- **Improved collaboration:** DevOps encourages collaboration between develop-

ers, testers, and IT operations teams, which can lead to improved communication, faster problem-solving, and better teamwork.

- **Increased agility:** DevOps enables organizations to respond more quickly to changing market conditions, customer needs, and technological advancements by allowing them to make changes to software more quickly and with greater efficiency.
- **Enhanced quality:** DevOps processes include continuous testing and integration, which can lead to higher-quality software releases with fewer bugs and errors.
- **Improved security:** DevOps practices include security testing and monitoring, which can help organizations identify and address security vulnerabilities in their software before they become a problem [47].
- **Cost savings:** By reducing manual processes and automating routine tasks, DevOps can help organizations save time and money.

Overall, DevOps can help organizations become more efficient, innovative, and responsive to changing business needs, which can lead to increased competitiveness and customer satisfaction. Such benefits are counterbalanced by challenges. Some of them are:

- **Cultural shift:** DevOps requires a cultural shift in how organizations approach software development, testing, and deployment. This can be a challenge, especially if the organization has a history of siloed teams and a lack of collaboration.
- **Toolchain complexity:** DevOps requires a range of tools and technologies, which can be complex to set up and manage. Organizations may need to invest in new infrastructure, tools, and training to support a DevOps culture.
- **Security concerns:** While DevOps can improve software security, it can also introduce new security risks if security is not integrated into the DevOps process from the beginning. Organizations must prioritize security throughout the DevOps process to ensure that software is secure and protected from threats.
- **Legacy systems:** Legacy systems may not be compatible with DevOps practices, which can create roadblocks to implementation. Organizations may

need to invest in modernizing legacy systems before implementing DevOps.

- **Lack of expertise:** DevOps requires specialized expertise in areas such as automation, testing, and continuous delivery. Organizations may need to invest in training or hiring new staff to build this expertise.
- **Resistance to change:** Resistance to change is a common challenge when implementing DevOps. Teams may be resistant to new processes, tools, and ways of working, which can slow down adoption and create roadblocks.

The last one in particular is a well-known challenge. Due to such resistance in the latest years multiple alternatives to DevOps have been proposed, some of them are AIOps, DevSecOps, and, last, MLOps [22, 70]. While some of them have been developed to simplify the development process, some others have been a clear evolution of the concept of DevOps. As an example, while AIOps uses artificial intelligence (AI) to simplify IT operations management and accelerate and automate problem resolution in complex modern IT environments [85], MLOps has been introduced to include the figure of the ML developer in parallel with the figure of the software developer.

2.1.2 Machine Learning

The field of data science has undergone several revolutions. Some of its subfields, such as image and signal processing and analysis, have undergone a transition from classical techniques, such as those based on signal transforms [68, 57], to AI and its subcategories.

Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) are related terms that are often used interchangeably, but they refer to different concepts and technologies in the field of Data Science (Figure 2.2).

AI refers to the broader concept of creating machines or systems that can perform tasks that typically require human intelligence, such as speech recognition, natural language processing, decision-making, and image recognition. AI includes various subfields among which there is ML.

ML involves training algorithms to recognize patterns in data and make predictions or decisions based on that data and patterns. Machine learning algorithms learn from historical data to make predictions or decisions without being explicitly programmed to do so. Machine learning is commonly used in applications such as image

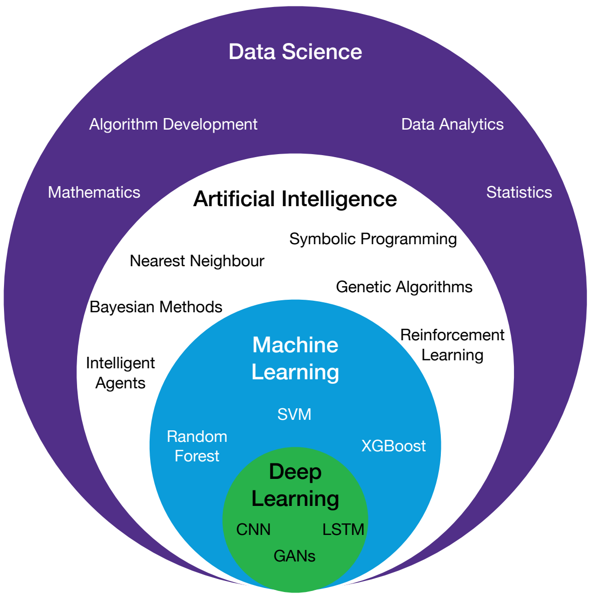


Figure 2.2 The differences between AI, ML, and DL in Data Science

recognition, natural language processing, and predictive analytics [56].

DL is a subset of ML that uses neural networks with multiple layers to recognize patterns in data. Deep learning algorithms learn from large volumes of data and can perform complex tasks such as speech recognition, image recognition, and (advanced) natural language processing [58]. Deep learning is often used in applications such as self-driving cars, recommendation systems, and facial recognition.

Libraries play a crucial role in AI and ML by providing pre-written code, algorithms, and tools that can be used to build and train machine learning models. These libraries save time and effort for developers, as they don't have to write everything from scratch. Some popular libraries used in AI and ML include TensorFlow, Keras, PyTorch, Scikit-learn, and Pandas. In addition, libraries provide access to vast amounts of data, which is essential for training and testing machine learning models. Many libraries also offer visualization tools that help developers and researchers understand the data and model results. Furthermore, libraries facilitate collaboration among developers and researchers by providing a shared foundation for building and testing AI and ML models. They also enable developers to keep up with the latest research and advancements in the field.

Most of such libraries are open source, and like OSS it is important to understand

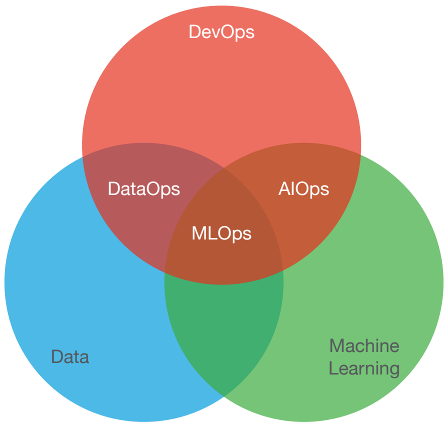


Figure 2.3 Different branches derived from DevOps. Reprinted with permission from PUBLICATION V.

what are the consequences of embedding open source components in the chosen development pipeline [24]. For example, a common problem in ML is the requirement to install a particular version of libraries due to Dependency Conflict [81]. One alternative might be to version lock your project but this will prevent the implementation of new features [93]. This is because many of these libraries may not be compatible with future updates. This can lead to incompatibilities and the inability of the older system version to work seamlessly with newer iterations. [45].

2.1.3 MLOps

The growing prevalence of software based on Machine Learning (ML) has led to the need for a new approach to develop software effectively. In parallel, there has been a growing need to apply DevOps practices to different areas of focus within the software development ecosystem. These areas of focus may include the development of ML-based software. To address these specific needs, new sets of practices have emerged that combine DevOps with the unique requirements of these areas.

The evolutions of DevOps aim at providing a tailored approach to software development that takes into account the specific challenges and considerations of these different entities. As the use of these specialized practices continues to grow, they are receiving increasing attention and becoming a key aspect of modern software development practices. Some of these evolutions are schematized in Figure 2.3.

As an example, DataOps, like its predecessors, does not have a clear definition among those proposed. The one that seems to include all its characteristics is the one proposed by Ereth, who defines DataOps as *"set of practices, processes, and technologies that combines an integrated and process-oriented perspective on data with automation and methods from agile software engineering to improve quality, speed, and collaboration and promote a culture of continuous improvement"* [30].

This definition aims at highlighting the importance of creating and versioning valuable data for every possible application, from data science [59, 35] to deep learning [60, 50].

The previously introduced concepts did not consider the development and deployment of applications based on the resulting data. Therefore, a separate set of practices have emerged that focus on the lifecycle management of data-driven applications. This set of practices is called MLOps.

MLOps as it provides a framework for managing the data used in ML projects. For this reason, MLOps is heavily dependent on the concepts proper of DataOps. As MLOps involves the entire machine learning lifecycle, from data preparation to model training, deployment, and maintenance, it is crucial to have a streamlined process for data management that ensures data quality, consistency, and reliability. In particular, the data must be representative of the environment and the problem that the ML-based system is trying to solve. However, as we live in a dynamic world, the model trained on a particular dataset may decay over time. When this happens, we may be faced with data drift (in some cases also defined as covariate shift) or concept drift [53, 105]. Data drift occurs when changes in the data lead to poor performance of the model, whereas concept drift occurs when the relationship between the data and the labels changes, meaning that the goal of the prediction changes. Whereas in the first case it is due to a different input, in the second case it is the interaction between the input and the output that is different. To mitigate this severe problem detectors have been proposed [80].

Different definitions focus on different stages and phases of MLOps. Among these definitions one of the most famous is the one from Google [18] which describes different levels of MLOps based on the automation level. Following their definition it is possible to start with a completely manual MLOps pipeline (defined as level 0) and, building on top of it by increasing the levels of automation it is possible to achieve a level 2 of automation which includes CI, CD, and Continuous Training (CT). On the opposite side, in [62] the focus has been placed on proposing an MLOps loop

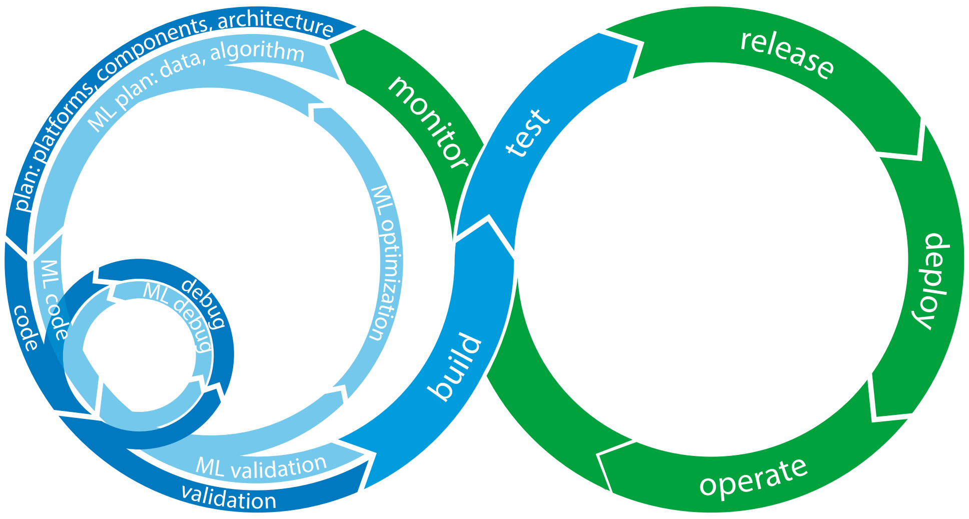


Figure 2.4 The MLOps Lifecycle infinite loop. Reprinted with permission from [62]. ©IEEE 2022.

that highlights the differences between the original DevOps loop (Figure 2.1) and the process of developing ML-based software. The proposed infinite loop is depicted in Figure 2.4.

From this, it is possible to see that the side of the loop related to Ops is mostly unchanged. Starting from the *Planning phase*, it encompasses all activities that occur prior to the development team beginning to write code, such as gathering requirements and designing the system's architecture. Specifically for ML, planning involves identifying the problem to be solved and everything that is related to the data (including preprocessing and filtering). This information is then used to determine the most appropriate data analysis methods, such as classification or regression, and to select suitable algorithms, such as Random Forest or Deep Learning. Additionally, supervised algorithms may require a data labeling step.

During the *Coding phase*, the development teams are responsible for implementing both the system's code and the machine learning algorithms. Once the code has been written, it undergoes local validation to ensure that it is functioning correctly. In traditional DevOps, the *Validation phase* typically involves running local tests to verify the code's functionality before committing it. In the context of ML, however, validation refers to evaluating the performance of the ML model using previously unseen data. If the validation process indicates that the ML-based approach is not appropriate for the data or algorithms, the development team needs to return to the planning phase to optimize the model for better problem-solving. Once the ML optimizations have been made and the system has been locally tested, the ML code is integrated into the system code. This integration ensures that the system is equipped with the necessary ML capabilities to perform its designated tasks effectively.

Following, starting from the *build phase*, everything is performed similarly to DevOps.

Another similarity with DevOps is the high exploitation of automation tools for accomplishing automation of the whole MLOps pipeline [11, 6, 16, 75]. Some works also concentrate on the integration of tools used in DevOps and MLOps [9, 39]. In [83], the authors conducted a Multivocal Literature Review (MLR) to accomplish two objectives, the identification of tools that enable and facilitate the development of MLOps pipelines, and the identification of their key characteristics and features in order to offer a comprehensive understanding of their significance. The work thoroughly investigated 13 MLOps tools and their main features which have been divided into 3 main categories: General Features, Data Management Features, and Model Management Features.

At a later stage, the work has been extended in [67] passing from 13 to 84 tools. In the work, the focus has been placed on the question: *which phase of the MLOps infinite pipeline each tool contributes to?* Together with the main phases of the MLOps pipeline, some of the tools have been covering more than one phase, starting from the End-to-End Full-Stack MLOps Tools (covering all the phases) to the different levels of automation phases (CT, CD, CI).

2.2 Software Architectures

Service-oriented architecture (SOA) is a software design approach that promotes the reusability of software components through service interfaces that use a common communication language over a network. In SOA, a service is an autonomous unit of software functionality, or a set of functionalities, created to accomplish a specific task such as retrieving specific information or performing an operation [31].

The earlier model of software development was referred to as *monolithic* since the entire code for the application was integrated into a single deployment. If there was any issue with a particular feature of the application, the entire system had to be temporarily shut down to address the issue and then redeployed as a new version.

To overcome such problem in SOA, services interact with each other through a system of loose coupling. This allows the components, or elements, of a system or network to exchange information or coordinate a business process without having strong dependencies between them. Some of the benefits achieved compared to the monolithic approach are: flexibility, scalability, lower costs, faster time to the market [98, 21].

A further step from monolith has been taken with the development of Microservices (MSs). Like SOA, MSs are based on loosely coupled connections with a higher degree of freedom. Such approach to development makes the software, web or mobile applications, a suite of independent services [32, 71]. The independent services composing an MS communicate between each other via APIs to build specific applications that perform business functionality in a more agile, scalable, and resilient manner.

As MLOps aims at building and deploying applications as microservices it is important to understand not only the importance of microservices for software architectures but also what is the relation between AI and microservices.

Different works have been studying the relationship between AI and Microservices. In [66], an SLR has been performed with the goal of investigating how AI has been used to support the design, development, and operations of MSs.

2.2.1 Computation Distribution

The hardware device that is currently receiving the most attention in relation to the further advancement of ML is the edge. In this thesis, we will refer to the edge as those devices which are capable of performing computation at the *edge* of the network [102]. This can be attributed to the reduced latency when providing real-time results compared to the cloud. The availability of devices capable of providing high storage and computation resources has inevitably reduced the necessity of being connected to centralized servers in the cloud. However, this is not always the case. In situations where the computational power available on the edge devices is not enough, the connection to the cloud cannot be avoided and the use of tools capable of distributing computational tasks between edge devices and cloud resources becomes vital.

We can separate those tools capable of moving the computation vertically (or edge-to-cloud tools) [25] from those capable only of moving the computation horizontally (or edge-only tools) [96]. Edge-to-cloud tools encompass a range of hardware, software, and services that are utilized to collect, process, and analyze data from various edge devices such as sensors and machines. The data collected by these devices is transmitted to cloud platforms for further processing and analysis. "A cloud-based system capable of sensing its environment, learning from it, and opportunistically and dynamically adapt its computational load as well as its outcome" is defined as a *Cognitive Cloud* [64].

With the increasing demand for real-time data processing and analysis, edge-to-cloud tools have emerged as a crucial technological solution. The combination of edge devices, cloud platforms, and software services has enabled the collection, processing, and analysis of data from distributed edge devices in real time. This has paved the way for enhanced data insights and analytics, enabling businesses to make more informed decisions based on real-time data. The availability of edge-to-cloud tools has transformed data processing and analysis from a centralized approach to a distributed approach, allowing for greater flexibility, scalability, and efficiency. By leveraging the power of edge-to-cloud tools, businesses can stay ahead of the competition and gain a competitive edge in the market.

In [69], a list of 40 different tools capable of performing offload, orchestration, or other computational tasks in the cloud continuum has been provided.

3 RESEARCH METHODS

This chapter presents the research methods used in the different publications.

3.1 Publication I - Survey

In Publication I [48] the main research method used to answer the different RQs has been a survey [43]. In particular, the work has been conducted in 3 main steps:

1. **Step 1:** Interviews among 23 experienced software developers and project managers have been performed to investigate the factors and metrics that have been influencing the selection of OSS, as well as the source of information
2. **Step 2:** Analysis of the APIs of the source of information identified
3. **Step 3:** Evaluation of the accessibility of metrics gathered in the preceding phases via public APIs of information sources that practitioners use, among a pool of 100k projects.

In the first step, we created and conducted a semi-structured interview using a questionnaire to investigate the factors considered by practitioners during the selection of OSS in the software product development process. The interviews utilized a questionnaire that was previously employed in other studies to identify the factors deemed significant for assessing OSS [12, 92]. To establish the profile of the respondents, we gathered demographic information pertaining to their experience with OSS. This included their years of experience in selecting OSS components for integration into the software they develop. Additionally, we collected relevant details about their roles, predominant experience, and company information such as the application domain, organizational size (number of employees), and the number of employees within the respondents' own team. The population identified for the interview was based on 4 main criteria:

1. Currently developing software

2. At least 5 years of software development experience
3. At least 3 years in the specific domain
4. At least 3 years of experience in OSS selection.

Once identified the population, we requested the respondents to compile a list of factors that are known to be generally taken into account when adopting OSS software for integration into the products they develop. They were then asked to rank these factors based on their perceived importance, using a scale ranging from 0 to 5. In this scale, a rating of 0 indicated the factor as "totally irrelevant," while a rating of 5 indicated the factor as "fundamental". The open-ended questions regarding the application domain, additionally reported factors, platforms used for information extraction, and metrics employed for factor evaluation were subjected to analysis through open and selective coding [101].

The complete list of questions has been reported in a replication package [84]. Due to time limitations and the inability to conduct face-to-face interviews at public events, interviewees were chosen using a convenience sampling approach, also referred to as Haphazard Sampling or Accidental Sampling [8]. Despite this, we made efforts to ensure diversity among the interviewees by inviting an equal number of developers from both large and medium-sized companies, representing various domains.

The responses were interpreted by extracting distinct sets of similar answers and organizing them according to their perceived similarity. Two authors employed the open coding methodology to manually generate a hierarchical set of codes based on all the transcribed answers. Any coding discrepancies were discussed and resolved between the authors before applying the axial coding methodology [101].

During the interviews, the respondents identified a total of nine distinct sources of information and portals that they typically consult when selecting OSS. In the second step, we manually analyzed the APIs of such portals looking for APIs that allowed us to assess the information needed to measure the factors reported by the interviewees. The initial analysis of all the portals and the search for specific pieces of information were conducted independently by the first two authors of the work. The obtained results were subsequently compared. In the event of any discrepancies, all the authors participated in comprehensive discussions to address and resolve any inconsistencies, ensuring a unanimous consensus was reached. In this step some of the factors were not directly analyzable while some of the metrics could not have been extracted.

The last step was divided into 3 substeps:

1. Project selection: The top 100K GitHub projects, based on the number of stars were selected.
2. Information extraction: Only the information needed to evaluate the factors was extracted (ex. for Popularity we extracted Number of Watch, Number of Stars, Number of Forks, Number of Downloads).
3. Analysis of the information available: Analysis of which information is actually available for each project.

The results obtained in this work, even if they are aimed at OSS, can also be applied to ML libraries, since they are generally categorized as open source.

3.2 Publication II - Model development

In Publication II [46] a model to assess the risk of abandonment for OSS components has been developed. The classic risk assessment notion is applied to calculate the abandonment risk of each component [14]. This involves evaluating the probability and impact of a risk event. When assessing the risk of embedding software we refer specifically to Qualitative and Quantitative Risk [49]. For Qualitative Risk, we refer to the probability that different risks will occur, while with Quantitative with the overall effect of the different risks. The abandonment risk of OSS components within a software system is determined by considering two main factors. Firstly, the likelihood of each component losing maintenance support during a specified period is taken into account. This likelihood can be influenced by several factors such as the component's age, popularity, level of activity in the community, and the number of contributors. By analyzing these factors, practitioners can estimate the probability of each component being abandoned (Qualitative Risk).

Secondly, the importance of each component for the main system is evaluated. This importance can be assessed based on factors such as the component's functionality, criticality to the system's operation, and ease of replacement. Components that are critical to the system's operation and are difficult to replace are considered more important and have a higher impact on the system's overall risk level (Quantitative Risk).

In the context of OSS components, the probability of an OSS component being abandoned represents the likelihood of a risk event, while the importance of the component for the main system reflects its impact on the system's overall risk level. By combining these two factors, practitioners can determine the abandonment risk of each component and prioritize their maintenance or replacement accordingly.

To mitigate the abandonment risk of OSS components, practitioners can implement various strategies such as monitoring the community activity around the component, seeking alternative components or solutions, contributing to the component's development, or developing custom solutions. By proactively managing the abandonment risk of OSS components, practitioners can minimize the impact of potential failures and disruptions in the software system.

The OSSARA process is depicted in Figure 3.1. Beginning with a software system that incorporates multiple OSS components (e.g. 14 components), the first step is to determine the abandonment probability and weight of each component within the given time frame. The abandonment probability is represented by a color code while the weight is denoted by the size of the box. These two pieces of information are then combined to calculate the overall risk of the software system being abandoned within the specified period. The overall abandonment risk R_a for a system that integrates k OSS components is calculated as:

$$R_a = \sum_{m=1}^{k} w(O_m) * r(O_m), \tag{3.1}$$

where $w(O_m)$ represents the weight of the OSS component O_m, while $r(O_m)$ the risk that O_m will be abandoned.

Predicting the risk of an OSS component being abandoned is a complex issue that involves multiple factors, such as poor performance and insufficient maintainability. Traditionally, the number of commits performed on the system repository within a specific time interval has been used as the sole predictor of abandonment. However, this approach oversimplifies the problem and fails to account for other critical factors that may influence the abandonment risk.

It is possible that an OSS community might not focus on committing, but the contributors remain active in handling pull requests and discussing relevant issues. In such cases, other measures such as the daily number of commits, issue comments and closed pull requests should be considered to assess the risk of abandonment. Thus,

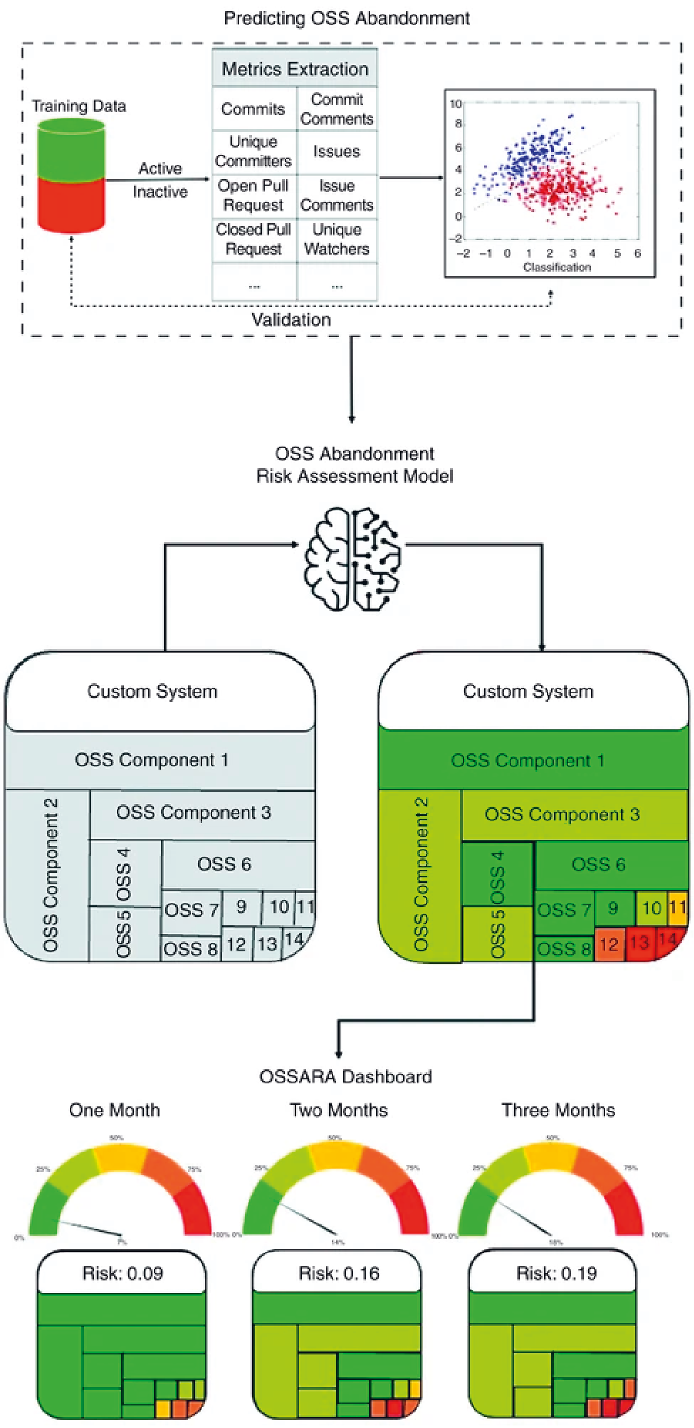


Figure 3.1 The OSSARA process. Reprinted with permission from PUBLICATION II.

predicting the abandonment risk of OSS components is a multi-concern assurance problem that requires a comprehensive approach to account for all the relevant factors.

The proposed model applies supervised techniques to predict the abandonment of OSS components. This could be achieved following a 3-step pipeline:

1. *Data crawling:* At first data from a selected source needs to be gathered. The metrics chosen for evaluation include the number of commits, commit comments, unique committers, issues, issue comments, watchers, and open/closed pull requests.

2. *Data preprocessing:* To generate training data for each OSS project, we adhered to the criteria established by our case company. We classified projects as "active" if they met the following criteria: 1) had more than $2,000$ commits, 2) had more than $1,000$ days of activity (from the creation day to the final commit day), 3) had at least one commit in the previous six months, and 4) had days with zero commits that account for less than 50% of the days of activity. We also ensured that the labeled training data aligned with the target prediction period (e.g., one, two, or three months) and that their dimensions were optimized to yield the highest accuracy possible.

3. *Data prediction:* We utilized the labeled and preprocessed data to train classifiers that exhibit optimal performance for the designated prediction periods. Once trained, the classifier was tasked with predicting whether a given OSS component was active or abandoned, based on the input data. The accuracy of the classifier was then used as an indicator of the probability that the OSS is either active or abandoned, as it reflects the likelihood of the prediction is correct.

The model has been validated by performing a preliminary evaluation on 12,208 OSS projects that contain at least 1,000 commits from at least five unique contributors and are watched by at least 100 users [97]. The dataset has been extracted from GHTorrent and labelled following the procedure described in the *Data Preprocessing* step. The classification described in the *Data Prediction* step has been performed using 4 different classification algorithms: decision tree, support vector machine, logistic regression, and naive Bayes.

3.3 Publication III - Systematic Mapping Study

In Publication III [65], the main research method used to define the concept of Cloud Continuum has been a systematic mapping study. The systematic mapping study has been carried out by taking into account the guidelines proposed by Petersen et al. [79].

The primary objective of a systematic mapping study is to systematically and impartially summarize and categorize the gathered information concerning the RQs. In this particular publication, the aim has been not only to describe all existing definitions of the "cloud continuum" and other pertinent concepts but also to examine the evolution of these definitions over time.

The approach encompassed four main steps. First, we established the research questions. Second, we formulated the search strategy. Third, we defined the data extraction strategy. Finally, we synthesized and visually presented the results obtained.

The objective of conducting a systematic mapping study is to establish a search query capable of retrieving a comprehensive collection of studies containing the desired definitions [42]. To achieve this, the search strategy entailed several steps, including defining the search string, identifying key sources, selecting primary studies, extracting data, and synthesizing the results.

The search strategy encompassed outlining the most relevant bibliographic sources and search terms, defining the criteria for inclusion and exclusion, as well as establishing a selection process to determine which studies should be included.

In this specific work, the search string has been:

(cloud AND (edge OR fog) AND continuum)

It has been performed on four bibliographic sources: Scopus[1], IEEEXplore Digital Library[2], the ACM Digital Library[3], and Web of Science[4]. The adoption of four databases ensured the completeness of the search results. The search has been conducted on March 1st, 2022, retrieving 378 non-duplicated papers from the four sources.

To identify the primary studies from the initial search results, specific inclusion

[1]Scopus, https://www.scopus.com.
[2]IEEEXplore Digital Library https://ieeexplore.ieee.org/.
[3]ACM Digital Library: https://dl.acm.org.
[4]Web of Science database: https://www.webofscience.com/.

and exclusion criteria need to be established. In this specific work, research papers published in journals or conferences that discussed the Cloud Continuum concept were included. Conversely, papers not in English, duplicates, unrelated to the defined research questions, non-peer-reviewed, as well as work plans, roadmaps, posters, and vision papers have been excluded.

With the inclusion and exclusion criteria in place, the selection of primary studies involved two steps. Initially, two authors independently reviewed the titles and abstracts of each paper to determine whether it met the criteria for exclusion or required further examination. In cases where there was disagreement between the two authors, a third person intervened to make the final decision. In this publication, out of the 378 papers screened, 93 had disagreements, resulting in a Cohen's kappa coefficient of 0.51, indicating a moderate level of agreement [28]. Consequently, 181 papers that needed to be considered for the subsequent step were identified. Subsequently, a snowballing process that involved incorporating all the papers referenced within the 181 identified papers was initiated. It involved applying the same inclusion and exclusion criteria to the titles and abstracts of the additional papers. As a result, we were able to include two more papers that met the criteria.

Then, based on the inclusion and exclusion criteria, the 6 authors of the work had to independently read and thoroughly evaluate each of the papers (183 in this particular work). The result of the evaluation process is the ultimately selected list of papers that met the criteria for further analysis defined as Selected Papers (SPs). The different SPs from this publication can be consulted in Appendix A of PUBLICATION III.

Relevant data from the selected papers that provided insights into our research questions was extracted. Specifically, the focus has been placed on extracting definitions related to the "continuum," the year of publication, and information regarding the continuation of the cloud. In addition, extracted information about the type of publication, such as whether it was a conference paper or a journal article, was extracted.

A qualitative analysis among the authors to identify similarities and differences among these definitions was conducted. To address the research questions, a collaborative coding process was employed.

From each paper, the definition was extracted and transcribed onto a Post-it note to address the first RQ. One author affixed each Post-it note onto a whiteboard while the other authors read the definitions proposed in the other papers. Through extensive discussion, the authors examined the similarities and differences among each definition, deciding whether to group them together or create new ones.

Subsequently, the authors rearranged the Post-it notes to reflect groups of similar definitions and their key differences. For each definition, the authors followed a consistent process to identify shared elements.

Finally, the authors used different colored markers to highlight the extensions of the continuum to the cloud, addressing the third RQ.

3.4 Publication IV - Algorithm Development

In Publication IV [63] a smart scheduling algorithm has been proposed and after validation, it has been deployed in production. Given two files related to Topology (TN) and to the temporal evolution of the traffic (TS), the developed algorithm is composed of 3 different contribution factors:

1. **Static Weight:** factor pertaining to information that will remain constant in the immediate future and is thus static over time. It is computed by considering the topology of the system. In particular, the static weight s_w of each location as a weighted sum of this different information is:

$$s_w = \alpha * W_E + \beta * W_L, \tag{3.2}$$

The weights W_E and W_L are computed based on different factors assigned to them, represented by α and β respectively. W_E captures the weight determined by the number of edges within a single location, while W_L represents the weight derived from the connections between two distinct locations. More in particular W_E for a specific edge location x_E is:

$$W_E(x_E) = \sum_{i=1}^{n} [l_i = x_E], \tag{3.3}$$

where l_i is the list of the unique edges, having the location as the prefix.

W_L is:

$$W_L(x_E) = \sum_{i=1}^{n} [lS_i = x_E | lD_i = x_E], \tag{3.4}$$

where lS_i is the list of all source locations and lD_i is the list of all destination locations.

2. **Dynamic Weight:** factor that encompasses all information that is subject to change over time and is thus linked to throughput between different nodes. More specifically, in this model, the Dynamic weight corresponds to the number of active connections each location has during different time slots.
3. **Cluster ID:** ID related to location-related clusters.

```
for $l_i$ in $TN$ do
    $W_E(x_E) = \sum_{i=1}^{n}[l_i = x_E]$;
    $W_L(x_E) = \sum_{i=1}^{n}[lS_i = x_E | lD_i = x_E]$;
    $s_w(x_E) = \alpha * W_E(x_E) + \beta * W_L(x_E)$;
end
$C = NetworkX(W_L)$
$DW = AssignDynamicWeights(TS)$
$SW = sort(sw)$                              ▷ From lowest to highest value of $x_E$
for $x_E$ in $SW$ do
    $TF(x_E) \leftarrow \max(DW(x_E))$
    if $SC(C(TF))$ is empty then
        $SC(C(TF)) = TF(x_E)$
    else
        $TF2(x_E) \leftarrow \max(DW(x_E), 2)$
        if $SC(C(TF2))$ is empty then
            $SC(C(TF2)) = TF2(x_E)$
        else
            $TF3(x_E) \leftarrow \max(DW(x_E), 3)$
            if $SC(C(TF3))$ is empty then
                $SC(C(TF)) = TF(x_E)$
            end
        end
    end
end
```

Algorithm 1: Smart Edge Provisioning Algorithm. Reprinted with permission from Publication IV. ©Springer Nature 2023.

The developed algorithm is summarized in algorithm 1. In more specific terms,

1. For each possible location (l_i) in TN, the Edge-Based weight (W_E) and the location-based weight (W_L) were calculated as described in Equation 3.3 and 3.4, respectively.

2. Once both weights are computed, the static weight s_w for each location is obtained.

3. Through W_L, the cluster numbers are determined using NetworkX.

4. For each specific location, the dynamic weights (DW) are assigned by identifying the minima (first, second, and third) in TS. This means that the time frame with the least amount of data sent will be assigned the highest dynamic weight (3), the second least will receive the second-highest dynamic weight (2), and so on until all weights are assigned.

5. s_w is sorted in ascending order. This approach prioritizes locations with fewer edges and connections, as they will have fewer opportunities for redirecting connections to adjacent edges, thus impacting fewer users.

6. For each location x_E in s_w, the maxima in DW are searched and the corresponding time frame ($TF(x_E)$) when the maxima occurs are identified.

7. For every location within the same cluster C, if the time frame is available, it is assigned to x_E. If not, the same procedure for the second and third maxima is repeated. If all the detected time frames have already been reserved, we move on to the next location.

Once all the locations were assigned, we had a clear schedule indicating which location should perform provisioning at each time frame (TF). Additionally, we had a list specifying the correct time frame for provisioning at each location. However, it is possible that some locations could not identify a suitable time frame. In such cases (usually less than 5% of locations), these locations could be assigned to empty time frames within their respective clusters (C) without causing any significant impact.

For model validation two different metrics were proposed. The intra-edge impact measured the algorithm's capability of preserving dense active connections during periods of high throughput, while also significantly penalizing situations where suggested scheduling was not feasible. On the other hand, as the proposed algorithm aimed at minimizing the amount of data lost during the provisioning through optimal scheduling, the traffic impact measured the ability of the algorithm to perform handovers.

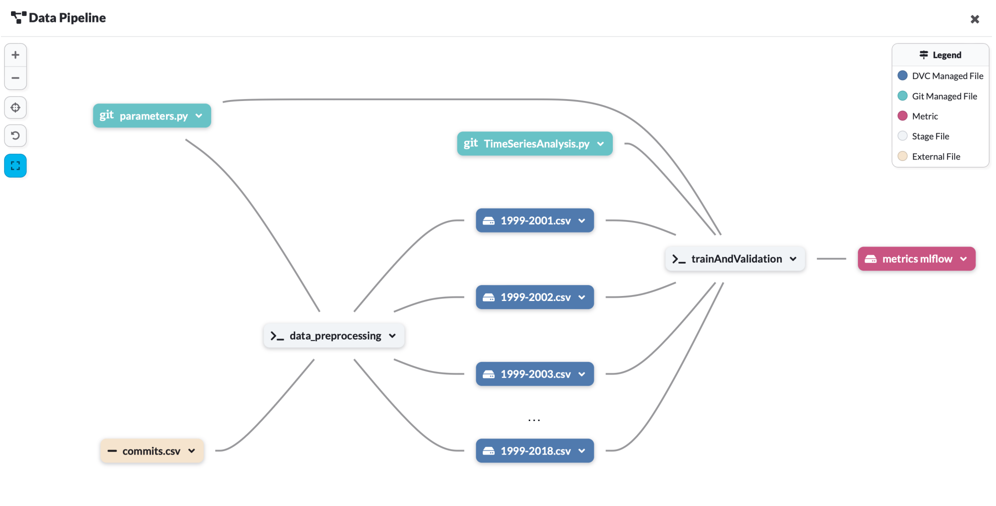


Figure 3.2 The data pipeline. Reprinted with permission from PUBLICATION V.

3.5 Publication V - MLOps Pipeline Creation

In PUBLICATION V [61] the model presented in PUBLICATION II has been developed following MLOps guidelines. The process of establishing an MLOps pipeline presents a distinct approach compared to both basic ML applications and classic software development.

In this case, the selection of tools used throughout the entire process becomes a crucial step that must be considered during the planning phase. Furthermore, as part of the planning phase, this tool selection might undergo variations as the development process unfolds.

The starting point of the work was the list of tools presented at first in [83] and then extended in [67], both based on the MLOps pipeline first presented in [62]. Following the list of MLOps tools, Dags-Hub, DVC, and MLFlow were selected as the tools for building this particular pipeline. As MLFlow is categorized as an "Ops" tool, DagsHub and DVC need to fulfill all the requirements for project development. Further, while both systems are classified as build tools, their collaboration can also encompass the planning phase, providing also a visual representation as shown in Figure 3.2.

One of the main objectives of this development was to enable a CT setup. To accomplish this, the dataset utilized in OSSARA [97] has been preprocessed, dividing it into distinct time frames.

The creation of these individual data frames was designed in a manner where the initial frame encompassed a span of 2 and a half years, serving as the basis for the first training. Subsequently, each subsequent frame extended the time period by an additional 3.5 years, resulting in a total of 18 different training sets.

The project began by initializing a DagsHub repository and configuring the DVC storage according to the instructions outlined in the DagsHub repository. As part of the planning phase of the MLOps pipeline, a *"dvc.yaml"* file was created to represent the data pipeline within the DagsHub repository.

For what concerns the Code/ML phase, the code was implemented in Python, specifically utilizing the scikit-learn library. The PyCharm integrated development environment (IDE) was chosen for development, and it was connected to the DagsHub repository to facilitate seamless commit and push operations.

Following, in the Build phase DagsHub and DVC were utilized. Since the application only requires an API, testing was handled by the deploying tool through automated testing prior to deployment. As for the data, since we are using a previously used dataset, there was no need for testing or performing drift analysis on it.

MLflow was utilized for all the Operations phases. Integration of MLflow was performed during the code development process, following the MLflow guidelines provided in the DagsHub repository.

After the development of the full MLOps pipeline, the automated process has been deployed. A complete training phase of a single dataset included four distinct algorithms. Once the training has been completed the ID from the best run was detected and used to perform a deployment based on REST API through MLflow.

4 RESULTS AND RESEARCH CONTRIBUTION

4.1 RQ1: What are the potential issues of embedding Open Source Libraries in MLOps?

PUBLICATION I - PUBLICATION II

In PUBLICATION I, we investigated and determined the criteria that practitioners presently take into account while choosing OSS, to pinpoint the sources (portals) that can be utilized for evaluating the aforementioned criteria, and to verify the viability of public APIs that enable the automated evaluation of those factors from the sources and portals. From the process described in Section 3.1, we identified 8 main factors and 46 sub-factors summarized in Table 4.1. In the table, RQ1 of PUBLICATION I focused on the number of times that a specific factor was mentioned by practitioners as well as the importance assigned by them, while RQ2 of PUBLICATION I reported the count of Metrics for each factor. The complete list of metrics can be consulted in the Appendix of PUBLICATION I.

More into detail, the most frequently mentioned factor by the interviewees is the License, which held a median importance rating of 4 out of 5. Interestingly, this was not the highest median importance value. Factors such as Community Support and Adoption, Performance, and Perceived Risk all received a median importance value of 4.5 out of 5. It's noteworthy that none of the participants discussed economic aspects and related sub-factors like license expenses or training costs.

Another important information that we aimed to extract in this work was which of the factor could be extracted automatically. To do this, previously in RQ3 of PUBLICATION I we extracted which sources of information and portals have been commonly used by practitioners. Once identified, a subset of the 4 most used portals have been used to extract the information needed to measure the factors through API among 100K projects in GitHub.

From the results of this work, it was emphasized the importance of having an OSS

RQ1			RQ2
Factor	#	Median	#Metrics
Community Support and Adoption	10	4.5	
Popularity	9	3	4
Community reputation	11	3	3
Community size	13	3	5
Communication	6	3.5	5
Involvement	9	3	1
Sustainability	11	3	1
Product Team	5	3	2
Responsiveness	1	5	1
Documentation	14	4	
Usage documentation	4	4	5
Software requirements	11	3	1
Hardware requirements	8	3.5	1
Software Quality Documentation	5	3	3
License	21	4	7
Operational SW Characteristics	6	4	
Trialability	5	3	2
Independence from other SW	11	3	4
Development language	5	4	3
Portability	1	4	1
Standard compliance	5	4	0
Testability	6	3.5	0
Maturity	6	3.5	11
Quality	6	3.5	
Reliability	3	4	6
Performances	4	4.5	1
Security	15	4	6
Modularity	3	3	1
Portability	3	4	2
Flexibility/Exploitability	3	3	3
Code Quality	13	4	6
Coding conventions	9	3	0
Maintainability	3	4	0
Testability	2	4	0
Existence of benchmark/test	4	3.5	4
Changeability	2	3.5	0
Update/Upgrade/Add-ons/Plugin	3	4	1
Architectural quality	5	3	0
Risk (Perceived risks)	7	4.5	
Perceived lack of confidentiality	5	1	0
Perceived lack of integrity	5	3	0
Perceived high availability	5	4	3
Perceived high structural assurance	5	2	0
Strategic risks	5	3	0
Operational risks	5	1	1
Financial risks	5	2	0
Hazard risks	5	4	5
Trustworthiness	6	4	
Component	4	3.5	3
Architecture	4	3	2
System	4	3.5	3
OSS provider reputation	4	3.5	0
Collaboration with other product	4	2.5	3
Assessment results from 3rd parties	2	3.75	0

Table 4.1 High-level factors considered during the adoption of OSS. Reprinted with permission from PUBLICATION I.

which is continuously maintained in order to the targeted level of stability, security, and quality. The risk of embedding software which might not be maintained and updated in the future could cause a heavy impact on the future of the developed software.

The potential consequences of the abandonment of OSS components cannot be overstated, as it can lead to a cascade of failures throughout a system, resulting in its complete inoperability. This "domino effect" can have severe implications, including financial losses, data breaches, and reputational damage. Moreover, the significance of this statement lies in the fact that in the event that an embedded software component is inaccessible or outdated, it can trigger a cascade of consequences throughout the entire system, potentially jeopardizing the overall functionality and security of the project. Therefore, it is crucial to maintain and regularly update OSS components to avoid such risks and ensure the system's reliability and performance. For this reason, in PUBLICATION II we introduced the OSS Abandonment Risk Assessment model (OSSARA).

The model introduced in section 3.2 is based on two main factor risks: the probability of each component losing maintenance support in a given time period, and the importance of each component to the main system, from which it is possible to compute the overall abandonment risk (Equation 3.1).

The objective of the model is to evaluate the likelihood of an OSS component being abandoned and its level of significance within a software system. By utilizing OSSARA, professionals have the ability to keep track of the system's risk level and decide whether to continue using or substitute OSS components.

The importance of reliability of specific libraries in ML is most important during the planning phase where the choice of the architecture to build is essential and during the coding phase where it is actually imported. This division provides clear similarity with the DevOps practices discussed in subsection 2.1.1, and provides the basis for the development of DevOps for ML-based software: MLOps.

Moreover, the model supports continuous adaptation and customization, enabling practitioners to achieve optimized predictions using current OSS activity data and selectively efficient, and even personalized algorithms. The model was requested by one company, which responded positively and incorporated it into its continuous integration/continuous deployment pipeline.

4.2 RQ2: What is the Cognitive Cloud Continuum?

PUBLICATION III

Since SOA has been adopted in cloud computing, the software development process has been significantly transformed [100]. The cloud is often considered an infinite resource pool on which applications can be developed and expanded for different objectives. Nevertheless, modern cloud systems are inherently complicated and may include a range of components and computational resources located at the network's edge, stretching from public to private cloud and potentially spanning across multiple regions.

The ability to scale up in cloud computing is achieved by creating and distributing multiple computing instances. Traditionally, containers have served as the basis for implementing architectures based on MSs. Nonetheless, the latest developments in serverless computing and Functions as a Service (Faas) underscore the role of the cloud as a platform that abstracts the underlying infrastructure resources [74, 5].

The widely accepted definition of Cloud Computing was coined by the National Institute of Standards and Technology (NIST) in 2011. According to such definition, Cloud Computing is *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."* [54].

The computation that happens at the edge of the network is defined as Edge Computing [103]. Due to its proximity to the user, it is characterized by short latency which makes it a better candidate for real-time applications in contrast to cloud computing where transmission of data, and allocation of resources typically includes delays. However, this quality comes with a price as edge devices do not present high computation capacities.

Everything that is between the Cloud and the Edge nodes is referred to as the Fog. Fog nodes aim to reduce the workload on the cloud and can host certain services that would normally be on the cloud, leading to faster response times and less network traffic to the cloud [17].

Edge and fog computing can provide advantages in terms of cost savings related to data transfer, storage, and processing for applications that require the processing of large amounts of data. Such applications can include, for instance, the processing

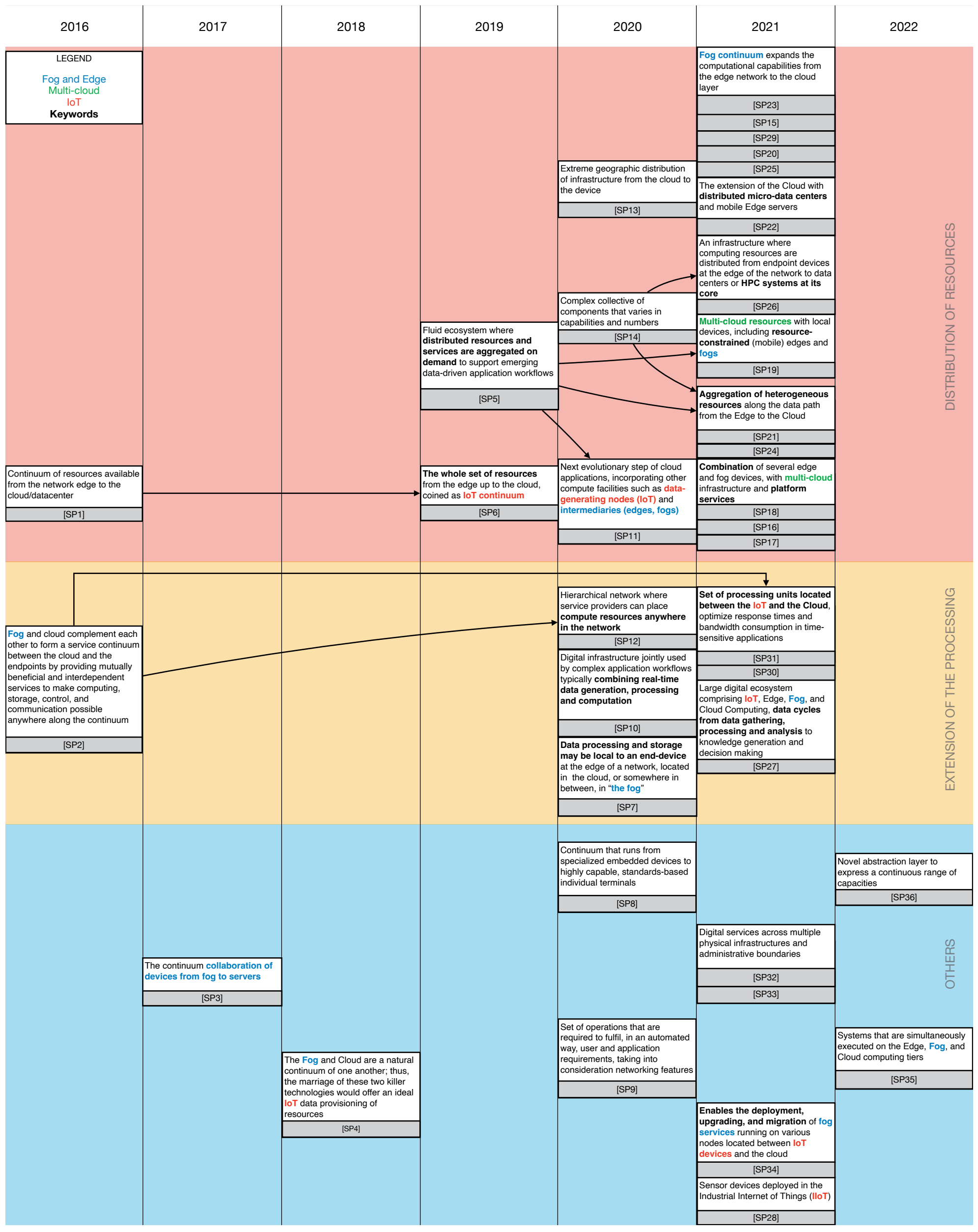

2016
2017
2018
2019
2020
2021
2022
LEGEND
Fog and Edge
Multi-cloud
IoT
Keywords
DISTRIBUTION OF RESOURCES
EXTENSION OF THE PROCESSING
OTHERS
Fog continuum expands the computational capabilities from the edge network to the cloud layer
[SP23]
[SP15]
[SP29]
[SP20]
[SP25]
Extreme geographic distribution of infrastructure from the cloud to the device
[SP13]
The extension of the Cloud with distributed micro-data centers and mobile Edge servers
[SP22]
An infrastructure where computing resources are distributed from endpoint devices at the edge of the network to data centers or HPC systems at its core
[SP26]
Complex collective of components that varies in capabilities and numbers
[SP14]
Multi-cloud resources with local devices, including resource-constrained (mobile) edges and fogs
[SP19]
Fluid ecosystem where distributed resources and services are aggregated on demand to support emerging data-driven application workflows
[SP5]
Aggregation of heterogeneous resources along the data path from the Edge to the Cloud
[SP21]
[SP24]
Continuum of resources available from the network edge to the cloud/datacenter
[SP1]
The whole set of resources from the edge up to the cloud, coined as IoT continuum
[SP6]
Next evolutionary step of cloud applications, incorporating other compute facilities such as data-generating nodes (IoT) and intermediaries (edges, fogs)
[SP11]
Combination of several edge and fog devices, with multi-cloud infrastructure and platform services
[SP18]
[SP16]
[SP17]
Hierarchical network where service providers can place compute resources anywhere in the network
[SP12]
Set of processing units located between the IoT and the Cloud, optimize response times and bandwidth consumption in time-sensitive applications
[SP31]
[SP30]
Fog and cloud complement each other to form a service continuum between the cloud and the endpoints by providing mutually beneficial and interdependent services to make computing, storage, control, and communication possible anywhere along the continuum
[SP2]
Digital infrastructure jointly used by complex application workflows typically combining real-time data generation, processing and computation
[SP10]
Large digital ecosystem comprising IoT, Edge, Fog, and Cloud Computing, data cycles from data gathering, processing and analysis to knowledge generation and decision making
[SP27]
Data processing and storage may be local to an end-device at the edge of a network, located in the cloud, or somewhere in between, in "the fog"
[SP7]
Continuum that runs from specialized embedded devices to highly capable, standards-based individual terminals
[SP8]
Novel abstraction layer to express a continuous range of capacities
[SP36]
Digital services across multiple physical infrastructures and administrative boundaries
[SP32]
[SP33]
The continuum collaboration of devices from fog to servers
[SP3]
Set of operations that are required to fulfil, in an automated way, user and application requirements, taking into consideration networking features
[SP9]
Systems that are simultaneously executed on the Edge, Fog, and Cloud computing tiers
[SP35]
The Fog and Cloud are a natural continuum of one another; thus, the marriage of these two killer technologies would offer an ideal IoT data provisioning of resources
[SP4]
Enables the deployment, upgrading, and migration of fog services running on various nodes located between IoT devices and the cloud
[SP34]
Sensor devices deployed in the Industrial Internet of Things (IIoT)
[SP28]

Figure 4.1 Definitions of cloud computing grouped by year and concepts. Each column represents a different year while the colored blocks represent different aspects. Arrows between two blocks indicate that there is a direct citation to the definition. Reprinted with permission from PUBLICATION III.

of data from thousands of sensors, audio, and video streams, as well as emerging ML-based solutions pushing the next stage of research in fields such as Computer Vision or Software Engineering.

In this scenario, it is hard to understand where to properly perform computation making the most of the available devices. As a consequence, in recent years, there has been a growing trend among researchers to concentrate on the concept of the cloud continuum paradigm, resulting in the presentation of several surveys and reviews on the topic [87, 76, 13, 72, 10, 82, 90, 4, 36, 40, 89, 77, 41]. However, its precise definition remains unclear, and the notion of cloud continuum is being described inconsistently in various papers.

The process performed in PUBLICATION III and described in Section 3.3 resulted in the identification of 36 SPs aiming at providing a valuable definition of the concept of the Cloud Continuum. The analysis performed investigated the evolution of the concept itself and categorized the works in the literature based on six different keywords that delineate the detected characteristics.

In 2016, the term "continuum" was first used to describe the area between the cloud and the edge, where computing, storage, control, and communication services could be provided. From 2017 to 2018, the term "continuum" continued to be used to describe the combination of fog and cloud and its potential to provide ideal Internet of Things (IoT) data provisioning. In 2019, the concept of the "computing continuum" was defined as a fluid ecosystem with aggregated resources and services, without emphasizing its positioning between the cloud and the edge. Also in 2019, the term "IoT continuum" was introduced and defined as the whole set of resources between the edge and the cloud.

Since 2020, more studies have provided definitions for the cloud continuum. Studies in 2020 placed the concept as the services between the cloud and the end devices (i.e., edge). In 2021, some studies mentioned fog as a critical entity in the definition of the continuum, and many of these studies anchored the continuum concept as the combination or aggregation of several fog, edge, IoT devices or services, or the extension of the cloud. Some studies also indicate that IoT is a crucial part of the cloud continuum concept, but the interpretation of the term differs slightly. Until February 2022, two studies also provided definitions for cloud continuum, one of which emphasized that it is a system simultaneously executed on the edge, fog, and cloud computing tiers, and the other defined it as a novel abstraction layer to express a continuous range of capacities.

During the evolution, the detected keywords were:

- **Multi-Cloud:** definitions that pertain to multiple entities within the cloud
- **Fog:** definitions specifically mentioning or referencing Fog
- **IoT:** definitions pertaining to the IoT and connected devices
- **Anywhere:** definitions explicitly reporting that the computation can be executed everywhere
- **Micro Datacenter:** definitions that clearly state the utilization of micro data centers with the aim of offering data processing and storage with low-latency access.
- **Simultaneous:** definitions that specifically mention the ability to perform computation on multiple nodes simultaneously.

These categories are reported by different color blocks in Figure 4.1 which maps the different SPs to the different categories. The different SPs can be consulted in Appendix A of PUBLICATION III.

The result of PUBLICATION III is a unified definition of Cloud Continuum as **"an extension of the traditional Cloud towards multiple entities (e.g., Edge, Fog, IoT) that provide analysis, processing, storage, and data generation capabilities"**.

4.2.1 Cognitive Cloud

Advancements in computer technology, big data, and Artificial Intelligence (AI) have led to the emergence of the term "cognitive" in the field of computer science [19]. Cognitive informatics, which originated in the early 21st century, investigates the inner workings of the brain and their application in computing and the IT industry [99]. In the 2010s, cognitive computing gained popularity as a research area, aiming to develop learning systems that can analyze vast amounts of data from various sources using a unified mechanism inspired by the human brain [55]. Cognitive computing employs algorithms rooted in cognitive science to enable machines to exhibit brain-like intelligence, particularly in processing and analyzing unstructured data [2]. This interdisciplinary approach, along with other technologies such as IoT, big data analysis, and cloud computing, holds significant potential for various applications [19].

Cloud computing plays a crucial role in supporting cognitive computing, which leverages shared and configurable computing resources on-demand [54, 3]. Cognitive computing, along with data mining and analysis, is predominantly conducted on cloud computing platforms. Moreover, cognitive computing can also enhance cloud computing in various domains. For example, edge cognitive computing architectures have been proposed to provide dynamic storage and computing services with cognitive capabilities at the network's edge [20]. Cognitive computing has also been utilized to detect attacks on IoT devices using fog-to-things computing [1]. However, despite the exploration of applying cognitive computing in cloud architectures, there is currently no established definition or theoretical framework for cognitive cloud. A definition has been proposed in [64], by means of a systematic mapping study similar to the one described in Section 3.3.

4.2.2 Cognitive Cloud Continuum

Following the outcomes of the previous sections, we can provide a unified definition for the COgnitive CLoud CONtinuum (COCLCON) by combining the previously introduced definitions. Specifically, we can inherit the *extension* proper of the Cloud Continuum PUBLICATION III and associate it with the *sensing, learning, and adapting* proper of the Cognitive Cloud [64]. We, therefore, define the COCLCON as:

Cognitive Cloud Continuum is an extension of the traditional Cloud towards multiple entities (e.g., Edge, Fog, IoT) capable of sensing its environment, learning from it, and opportunistically and dynamically adapting its computational load as well as its outcome.

The Cognitive Cloud Continuum revolution has already started. Researchers and practitioners are already starting to evolve their SDLC to cope with the requirements and needs of this new concept to take their implementations to the next level [95].

4.3 RQ3: How does the software development process change when doing MLOPs?

PUBLICATION IV - PUBLICATION V

To answer this research question, two different use cases have been developed. The first one in PUBLICATION IV is defined as a *static case* as an intelligent scheduler for

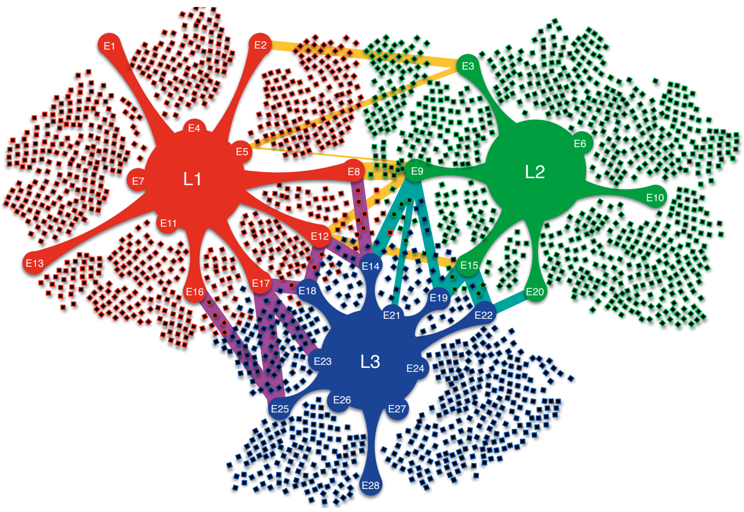


Figure 4.2 Example of a system with multiple *locations* (L1, L2, and L3), each with multiple *edge nodes* (E1, E2, ..), and a variety of *IoT devices* connected to the the edge nodes of the closest location (squares with same colors as locations). Moreover, the lines connecting edge nodes of different locations indicate the possibility to handover the connections of the IoT devices. Reprinted with permission from Publication IV. ©Springer Nature 2023.

updating edge services has been implemented using available data, while the second one, as it is based on an MLOps pipeline, implements a CT and can therefore be defined as a *dynamic case*.

4.3.1 Static Case

In Publication IV an intelligent scheduler for updating edge services within a SOA has been designed. The research involved examining a scenario where a significant Nordic enterprise operated a service-oriented system on edge nodes, delivering services to 270K IoT devices; an example of such a system is depicted in Figure 4.2. The scheduler, designed to reduce downtime during updates, recommended the most optimal update schedule that minimizes disruptions to connections for IoT devices.

The environment of the SOA within such an enterprise followed an agile methodology for the continuous development of the system, which necessitated deploying a fresh code version daily. However, deploying the new version entailed restarting every location, a process that took roughly 30 minutes on average, and had a significant impact on all connected IoT devices and related services provided by edge nodes. During this period, end-users connected to edge nodes in the location must be redirected to another edge node in a different location to minimize the number of service calls that are dropped. This process was complicated by the fact that IoT devices can only access neighboring locations due to the wireless technology employed. With the number of nodes and the daily upgrade time frame, it was not feasible to follow a sequential upgrade schedule as it would take over 405 hours (around 16 days).

The algorithm developed has been presented in Section 3.4. After deployment, the organization successfully achieved the capability to deploy fresh updates continuously, with only a single daily disruption of 30 minutes, resulting in a 20% reduction in service API calls.

4.3.2 Dynamic Case

Compared to the previous case, in Publication V the development of an MLOps pipeline is described. The goal of the work was to show what to consider when choosing the tools for each step of a pipeline. To create a valuable use case the OSSARA model from Section 3.1 has been developed using OSS tools forming an MLOps pipeline. One of the goals of the work was to perform CT, to this goal, the original dataset used in Publication II has been preprocessed and divided in order to simulate the act of continuously retrieving and creating data for the CT.

The creation of the pipeline presented in Section 3.5 highlighted the importance of data when developing an MLOps pipeline which is a direct consequence of the evolution from DataOps and the possibility to easily build and deploy ML-based software using MLflow.

In particular, the versioning of data increases the replicability and traceability of the work. The use of DVC in collaboration with DagsHub takes care of all that is necessary for data storage, versioning, and also for data planning. The simplified deployment provided by MLFlow through REST API not only allows to make any device to work as a server but also simplifies and speeds up the act of building and deploying the ML-based code. This means that in the COCLCON any device with enough computational power to build the network and perform the deployment could retrieve the weights from previous training and act as a server. More importantly, this does not mean the training needs to be performed on such a device allowing a computational offloading on a more powerful device.

In the work, the tools employed were kept to a minimum to ensure a straightforward yet comprehensive MLOps pipeline. It is worth noticing that the current state of tool development relies heavily on the intended application to be developed, thus smaller pipelines can be developed for different use cases.

5 DISCUSSION

This Chapter presents the outcomes of this thesis, therefore answers the Research Questions presented in Section 1.1 and summarized in Table 5.1. Based on such outcomes, the discussion is presented.

5.1 Answers to research questions

RQ$_1$ - What are the potential issues of embedding Open Source Libraries in MLOps? In order to answer RQ1, the Open Source nature of the different libraries composing ML-based code, and therefore MLOps, has been analyzed. First, the criteria that practitioners take into account while choosing OSS have been investigated (PUBLICATION I). Once the targeted OSSs have been identified it is important to be sure that such OSS is maintained and regularly updated to maintain its level of reliability and performance. The aim of PUBLICATION II is to provide a model for assessing the risk of embedding OSS (in this case libraries) into their software system, to help practitioners decide whether or not to select a particular OSS. The potential issues of embedding Open Source Libraries can be analyzed by making use of the proposed model, which can be used to measure the risk of embedding open source libraries when developing ML-based software, and thus MLOps pipelines.

RQ$_2$ - What is the Cognitive Cloud Continuum? For the goal of answering RQ2, a clear definition of the concepts of Cloud Continuum was identified (PUBLICATION III). Following, such definition has been connected to another definition in literature [64] to clearly define what is the Cognitive Cloud Continuum. This resulted in defining the Cognitive Cloud Continuum as: **"an extension of the traditional Cloud towards multiple entities (e.g., Edge, Fog, IoT) capable of sensing its environment, learning from it, and opportunistically and dynamically adapting its computational load as well as its outcome".** Such a definition has been provided to clearly state the main characteristics required for the elements composing such an

RQ1: What are the potential issues of embedding Open Source Libraries in MLOps?
OSSARA model to measure the risk of embedding open source libraries when developing ML-based software.
RQ2: What is the Cognitive Cloud Continuum?
Definition of the Cognitive Cloud Continuum.
RQ3: How does the software development process change when doing MLOps?
Use of MLops tools to perform **Continuous Training, data versioning and full automation.**

Table 5.1 Research Questions summary: RQs and the results achieved.

environment.

RQ$_3$ - How does the software development process change when doing MLOps? To answer RQ3 we developed two use cases validating the research previously performed. This is because, in the process of building the MLOps pipeline, it is essential not only to consider the risk associated with incorporating open source libraries during the coding phase, as specified in RQ1, but also to align with the criteria outlined in RQ2, relating to the development of a dynamic system that can be retrained and redeployed opportunistically.

In particular, 2 different use cases have been developed, the first has been based on the development of a single AI algorithm (PUBLICATION IV), while the peculiarity of the last use case has been the development of a model based on continuous training and therefore following the MLOps guideline (PUBLICATION V).

The first work has been an example of a scheduling system where multiple edge devices connected were receiving the time at which the update was starting. The developed system was taking care of the scheduling and the other devices were only receiving the information, therefore, the edge devices were not actively participating in the process. For this reason, the use case has been classified as *"static"*.

On the contrary, the dynamic case has been specifically tailored to compare how the process of Continuous Training, Continuous Integration, and Continuous Development influence software development. For this goal, the use of MLOps tools has been vital. More importantly, the use case selected for replication with MLOps as a software development process is a pre-developed use case with no automation. In developing the MLOps pipeline, special emphasis was placed on the process of filtering, preprocessing, and versioning the data, creating a first major difference from the previous model. Following, even if both systems have been developed in Python, the use of the different tools has allowed an increased knowledge of the different available models based on the different versions of the data.

5.2 Discussion of the results

To summarize, the results obtained in this thesis and in the publications from which they are derived, show the need for guidelines when developing ML-based software. Firstly, it is important to understand the risk of choosing one library over another. Highly available alternatives are not always positive, as some libraries may not be maintained and therefore may not be compatible with future updates of other embedded libraries. This will inevitably lead to problems and will make it impossible to deploy the software. In this particular situation, the presence of models such as OSSARA (Publication II), which have the ability to continuously monitor the risk associated with the inclusion of certain libraries, becomes fundamental.

On a different aspect, the different definitions focus on different aspects of MLOps creating confusion. The best example is to compare the guidelines provided by Google in [18] focusing on the levels of automation and other definitions such as [91] focusing on adding more stages to the DevOps pipeline. This thesis agrees on the importance of the automation proposition of [18] and the importance of including an ML-related part in the infinite DevOps loop.

The increased levels of automation provided by the use of MLOps tools and the possibility of connecting them into forming fully automated pipelines have been shown as a promising yet already established approach. On the same side, the evolution of the concept of the Cognitive Cloud Continuum has allowed the creation of environments based on different devices capable of communicating at different levels from the Cloud to the far edge. The latter, however, even if being mentioned in the cloud computing domain and funding agencies, did not have a clear definition or a set of interpretations agreeing on common characteristics as happened for DevOps.

The environment theorized in the definition of the Cognitive Cloud Continuum could potentially create powerful ecosystems capable not only of ensuring a more thoughtful use of computational power but also ensuring the ability of adapting to different circumstances within the environment such as data acquisition from different devices, reacting to anomalies in the data or reacting to devices unavailability.

The two concepts of MLOps and Cognitive Cloud Continuum have been shown as not only capable of coexisting but also benefiting from each other. This is because whenever a device in the Cognitive Cloud Continuum would not have the computational power to train a model, it could still exploit other devices in the environment and retrieve the weights of the model to perform the deployment at a later stage.

The use case provided in (Publication V) is a clear example of the possible improvements that could be achieved when using MLOps guidelines and pipelines over static ML-based applications. The proposed system has been a simulation of continuous data crawling, which is a continuous generation of data. According to the guidelines provided, this continuous flow of data needs to be versioned in the same way that code is versioned in DevOps. Subsequently, the data flow allows the system to perform continuous training, providing new versions of the model that could lead to more accurate models. These models can be compared at any stage, allowing the model with the highest selected metric to be automatically deployed, thus achieving full automation.

However, such full automation must be monitored on the basis of the application for which it is defined, proving that the 10 levels of automation (LOA) theorized in 1999 are increasingly relevant [29].

5.3 Threats to Validity

The work is subject to threats to validity. First of all, it is important to state those threats due to the empirical studies performed in this thesis. To this extent, the outcomes of the literature reviews could potentially be influenced by diverse factors of bias or error, encompassing inaccuracies in data extraction, and subjectivity in defining and implementing inclusion and exclusion criteria.

Following it is also important to separate those threats related to the development of ML-based software and those related to the Cognitive Cloud Continuum. In the first case, the main threat to validity is related to the generality of the study made. In order to give the best possible overview of the different phases composing the MLOps as well as the tools contributing to automating them the work does not focus on a single application (such as Computer Vision). For a particular application, a specific tool or a pipeline composed of a group of tools might be more efficient than a combination used for a different application (such as the use case provided in Publication V).

In the second case, the main threat to validity is the fast-evolving state of the art for task offloading. Adapting the software dynamically requires an environment such as the one theorized in the Cognitive Cloud Continuum, in its absence an orchestrator becomes necessary in the system and its presence makes the offloading nothing more than a sophisticated orchestration mechanism. The increased awareness in the Cognitive Cloud Continuum itself will allow the refinement of already existing algorithms and the development of new AI-based algorithms.

6 CONCLUSION

MLOps and the Cognitive Cloud Continuum are two concepts that are receiving increased interest nowadays. One commonality they share is the absence of a precise definition and a definitive list of tools that can be utilized.

This thesis explores and compares these two concepts to promote their widespread adoption in software development practices. At first, the importance of Open Source Libraries when developing ML-based code is explored. Following, the differences between DevOps and MLOps are emphasized to provide a vision of the different stages of machine learning-based software development and the role of the machine learning developer.

In the second Research Question, a clear definition of the concept of the Cognitive Cloud Continuum is provided.

The results from the different works composing this thesis have been validated through the development of use cases with the goal of showing differences in the process of software development before and after the development of MLOps guidelines.

This thesis contributes to the state of the research by increasing awareness of the concept of Cognitive Cloud Continuum and emphasizing the importance of following MLOps guidelines when developing ML-based software.

From the analysis performed in this work, it is possible to state that in the future the MLOps practices will become the new de-facto standard for software development. Moreover, the need to have such applications always available and on any device will make it necessary for the development of architectures supporting the Cognitive Cloud Continuum. The research performed will also allow practitioners and researchers to confidently select the proper set of tools when developing smart environments.

REFERENCES

[1] Abebe Abeshu and Naveen Chilamkurti. “Deep learning: the frontier for distributed attack detection in fog-to-things computing”. In: *IEEE Communications Magazine* 56.2 (2018), pp. 169–175.

[2] Ana Paula Appel, Heloisa Candello, and Fábio Latuf Gandour. “Cognitive computing: Where big data is driving us”. In: *Handbook of Big Data Technologies*. Springer, 2017, pp. 807–850.

[3] Michael Armbrust, Armando Fox, et al. *Above the clouds: A berkeley view of cloud computing*. Tech. rep. Technical Report UCB/EECS-2009-28, Uni. of California, 2009.

[4] Muhammad Asim, Yong Wang, Kezhi Wang, and Pei-Qiu Huang. “A review on computational intelligence techniques in cloud and edge computing”. In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 4.6 (2020), pp. 742–763.

[5] Mohammad S. Aslanpour, Adel N. Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. “Serverless Edge Computing: Vision and Challenges”. In: *2021 Australasian Computer Science Week Multiconference*. Dunedin, New Zealand: Association for Computing Machinery, 2021.

[6] Harvinder Atwal. *Practical DataOps: Delivering agile data science at scale*. Springer, 2019.

[7] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A software architect’s perspective*. Addison-Wesley Professional, 2015.

[8] Michael Battaglia, N Sampling, and PJ Lavrakas. “Encyclopedia of survey research methods”. In: *Publication date* (2008).

[9] Denis Baylor, Kevin Haas, Konstantinos Katsiapis, Sammy Leong, Rose Liu, Clemens Menwald, Hui Miao, Neoklis Polyzotis, Mitchell Trott, and Martin Zinkevich. "Continuous Training for Production ML in the TensorFlow Extended (TFX) Platform". In: *2019 USENIX Conference on Operational Machine Learning (OpML 19)*. Santa Clara, CA: USENIX Association, May 2019, pp. 51–53. ISBN: 978-1-939133-00-7. URL: https://www.usenix.org/conference/opml19/presentation/baylor.

[10] Malika Bendechache, Sergej Svorobej, Patricia Takako Endo, and Theo Lynn. "Simulating resource management across the cloud-to-thing continuum: A survey and future directions". In: *Future Internet* 12.6 (2020), p. 95.

[11] Benjamin Benni, Mireille Blay-Fornarino, Sébastien Mosser, Frédéric Précisio, and Günther Jungbluth. "When DevOps Meets Meta-Learning: A Portfolio to Rule them all". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 2019, pp. 605–612. DOI: 10.1109/MODELS-C.2019.00092.

[12] Vieri del Bianco, Luigi Lavazza, Sandro Morasca, and Davide Taibi. "Quality of Open Source Software: The QualiPSo Trustworthiness Model". In: *Open Source Ecosystems: Diverse Communities Interacting*. Ed. by Cornelia Boldyreff, Kevin Crowston, Björn Lundell, and Anthony I. Wasserman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 199–212. ISBN: 978-3-642-02032-2.

[13] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. "The internet of things, fog and cloud continuum: Integration and challenges". In: *Internet of Things* 3 (2018), pp. 134–155.

[14] Barry Boehm. "Software risk management". In: *European Software Engineering Conference*. Springer. 1989, pp. 1–19.

[15] Barry Boehm and Chris Abts. "COTS integration: plug and pray?" In: *Computer* 32.1 (1999), pp. 135–138. DOI: 10.1109/2.738311.

[16] Matthias Boehm, Arun Kumar, and Jun Yang. "Data management in machine learning systems". In: *Synthesis Lectures on Data Management* 11.1 (2019), pp. 1–173.

[17] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012, pp. 13–16.

[18] Google - Cloud Architecture Center. *MLOps: Continuous delivery and automation pipelines in machine learning*. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning. 2023.

[19] Min Chen, Francisco Herrera, and Kai Hwang. "Cognitive computing: architecture, technologies and intelligent applications". In: *Ieee Access* 6 (2018), pp. 19774–19783.

[20] Min Chen, Wei Li, Giancarlo Fortino, Yixue Hao, Long Hu, and Iztok Humar. "A dynamic service migration mechanism in edge cognitive computing". In: *ACM Transactions on Internet Technology (TOIT)* 19.2 (2019), pp. 1–15.

[21] Rui Chen, Shanshan Li, and Zheng Li. "From Monolith to Microservices: A Dataflow-Driven Approach". In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. 2017, pp. 466–475.

[22] Yingnong Dang, Qingwei Lin, and Peng Huang. "AIOps: Real-World Challenges and Research Innovations". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 2019, pp. 4–5.

[23] Sourojit Das. *DevOps Lifecycle : Different Phases in DevOps*. https://www.browserstack.com/guide/devops-lifecycle. 2023.

[24] Malinda Dilhara, Ameya Ketkar, and Danny Dig. "Understanding Software-2.0: A Study of Machine Learning Library Usage and Evolution". In: *ACM Trans. Softw. Eng. Methodol.* 30.4 (July 2021). ISSN: 1049-331X. DOI: 10.1145/3453478. URL: https://doi.org/10.1145/3453478.

[25] Alexander Droob, Daniel Morratz, Frederik L. Jakobsen, Jacob Carstensen, Magnus Mathiesen, Rune Bohnstedt, Michele Albano, Sergio Moreschini, and Davide Taibi. "Workrs: Fault Tolerant Horizontal Computation Offloading". In: *IEEE International Conference on Edge Computing (EDGE)* (2023). In press.

[26] Andrej Dyck, Ralf Penners, and Horst Lichter. "Towards definitions for release engineering and DevOps". In: *2015 IEEE/ACM 3rd International Workshop on Release Engineering*. IEEE. 2015, pp. 3–3.

[27] Geoffrey Elliott. *Global business information technology: an integrated systems approach*. Pearson Education, 2004.

[28] Khaled El Emam. "Benchmarking Kappa: Interrater agreement in software process assessments". In: *Empirical Software Engineering* 4.2 (1999), pp. 113–133.

[29] Mica R. Endsley and David B. Kaber. "Level of automation effects on performance, situation awareness and workload in a dynamic control task". In: *Ergonomics* 42.3 (1999), pp. 462–492. DOI: 10.1080/001401399185595. URL: https://doi.org/10.1080/001401399185595.

[30] Julian Ereth. "DataOps-Towards a Definition." In: *LWDA* 2191 (2018), pp. 104–112.

[31] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 1900.

[32] Martin Fowler and James Lewis. *Microservices*. http://martinfowler.com/articles/microservices.html. accessed: 2023-03-16. 2014. (Visited on 03/16/2023).

[33] Breno B Nicolau de França, Helvio Jeronimo, and Guilherme Horta Travassos. "Characterizing DevOps by hearing multiple voices". In: *Proceedings of the 30th Brazilian Symposium on Software Engineering*. 2016, pp. 53–62.

[34] Alfonso Fuggetta. "Open source software––an evaluation". In: *Journal of Systems and Software* 66.1 (2003), pp. 77–90. ISSN: 0164-1212. DOI: https://doi.org/10.1016/S0164-1212(02)00065-1. URL: https://www.sciencedirect.com/science/article/pii/S0164121202000651.

[35] Filipe Gama, Sergio Moreschini, Ilmari Huttu-Hiltunen, Olli Suominen, Robert Bregovic, and Atanas Gotchev. "CIVIT dataset: Stereoscopic 3D-360 videos of typical media production use cases". English. In: *European Light Field Imaging (ELFI) Workshop*. European Light Field Imaging Workshop ; Conference date: 04-06-2019 Through 06-06-2019. 2019.

[36] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A Rahmanian. “Resource management approaches in fog computing: a comprehensive review”. In: *Journal of Grid Computing* 18.1 (2020), pp. 1–42.

[37] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. “What is DevOps? A Systematic Mapping Study on Definitions and Practices”. In: *Proceedings of the Scientific Workshop Proceedings of XP2016*. XP ’16 Workshops. Edinburgh, Scotland, UK: Association for Computing Machinery, 2016. ISBN: 9781450341349. DOI: 10.1145/2962695.2962707. URL: https://doi.org/10.1145/2962695.2962707.

[38] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. “What is DevOps? A systematic mapping study on definitions and practices”. In: *Proceedings of the Scientific Workshop Proceedings of XP2016*. 2016, pp. 1–11.

[39] Theofilos Kakantousis, Antonios Kouzoupis, Fabio Buso, Gautier Berthou, Jim Dowling, and Seif Haridi. “Horizontally scalable ml pipelines with a feature store”. In: *Proc. 2nd SysML Conf., Palo Alto, USA*. 2019.

[40] Janis Kampars, Dainis Tropins, and Ralfs Matisons. “A Review of Application Layer Communication Protocols for the IoT Edge Cloud Continuum”. In: *2021 62nd International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS)*. IEEE. 2021, pp. 1–6.

[41] Puneet Kansal, Manoj Kumar, and Om Prakash Verma. “Classification of resource management approaches in fog/edge paradigm and future research prospects: a systematic review”. In: *The Journal of Supercomputing* (2022), pp. 1–60.

[42] B. Kitchenham and S Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. 2007.

[43] Barbara A. Kitchenham and Shari L. Pfleeger. “Personal Opinion Surveys”. In: *Guide to Advanced Empirical Software Engineering*. Ed. by Forrest Shull, Janice Singer, and Dag I. K. Sjøberg. London: Springer London, 2008, pp. 63–92. ISBN: 978-1-84800-044-5. DOI: 10.1007/978-1-84800-044-5_3. URL: https://doi.org/10.1007/978-1-84800-044-5_3.

[44] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. "A Survey of DevOps Concepts and Challenges". In: *ACM Comput. Surv.* 52.6 (Nov. 2019). ISSN: 0360-0300. DOI: 10.1145/3359981. URL: https://doi.org/10.1145/3359981.

[45] Valentina Lenarduzzi, Francesco Lomio, Sergio Moreschini, Davide Taibi, and Damian Andrew Tamburri. "Software Quality for AI: Where We Are Now?" In: *Software Quality: Future Perspectives on Software Engineering Quality*. Ed. by Dietmar Winkler, Stefan Biffl, Daniel Mendez, Manuel Wimmer, and Johannes Bergsmann. Cham: Springer International Publishing, 2021, pp. 43–53.

[46] Xiaozhou Li, Sergio Moreschini, Fabiano Pecorelli, and Davide Taibi. "OSSARA: Abandonment Risk Assessment for Embedded Open Source Components". In: *IEEE Software* 39.4 (2022), pp. 48–53. DOI: https://doi.org/10.1109/MS.2022.3163011.

[47] Xiaozhou Li, Sergio Moreschini, Zheying Zhang, Fabio Palomba, and Davide Taibi. "The anatomy of a vulnerability database: A systematic mapping study". In: *Journal of Systems and Software* 201 (2023), p. 111679.

[48] Xiaozhou Li, Sergio Moreschini, Zheying Zhang, and Davide Taibi. "Exploring factors and metrics to select open source software components for integration: An empirical study". In: *Journal of Systems and Software* 188 (2022), p. 111255. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2022.111255.

[49] Nguyen Duc Linh, Phan Duy Hung, Vu Thu Diep, and Ta Duc Tung. "Risk Management in Projects Based on Open-Source Software". In: *Proceedings of the 2019 8th International Conference on Software and Computer Applications*. ICSCA '19. Penang, Malaysia: Association for Computing Machinery, 2019, pp. 178–183. ISBN: 9781450365734. DOI: 10.1145/3316615.3316648. URL: https://doi.org/10.1145/3316615.3316648.

[50] Francesco Lomio, Diego Martínez Baselga, Sergio Moreschini, Heikki Huttunen, and Davide Taibi. "RARE: A Labeled Dataset for Cloud-Native Memory Anomalies". In: *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation*. MaLTeSQuE 2020. Virtual, USA: Association for Computing Machinery, 2020,

pp. 19–24. ISBN: 9781450381246. DOI: 10.1145/3416505.3423560. URL: https://doi.org/10.1145/3416505.3423560.

[51] Francesco Lomio, Sergio Moreschini, and Valentina Lenarduzzi. "A Machine and Deep Learning Analysis among SonarQube Rules, Product, and Process Metrics for Fault Prediction". In: *Empirical Softw. Engg.* 27.7 (Dec. 2022). ISSN: 1382-3256. DOI: 10.1007/s10664-022-10164-z. URL: https://doi.org/10.1007/s10664-022-10164-z.

[52] Lucy Ellen Lwakatare, Terhi Kilamo, Teemu Karvonen, Tanja Sauvola, Ville Heikkilä, Juha Itkonen, Pasi Kuvaja, Tommi Mikkonen, Markku Oivo, and Casper Lassenius. "DevOps in practice: A multiple case study of five companies". In: *Information and Software Technology* 114 (2019), pp. 217–230. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2019.06.010. URL: https://www.sciencedirect.com/science/article/pii/S0950584917302793.

[53] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. "Matchmaker: Data Drift Mitigation in Machine Learning for Large-Scale Systems". In: *Proceedings of Machine Learning and Systems*. Ed. by D. Marculescu, Y. Chi, and C. Wu. Vol. 4. 2022, pp. 77–94. URL: https://proceedings.mlsys.org/paper_files/paper/2022/file/1c383cd30b7c298ab50293adfecb7b18-Paper.pdf.

[54] Peter Mell, Tim Grance, et al. *The NIST definition of cloud computing*. Tech. rep. Computer Security Division, Information Technology Laboratory, 2011.

[55] Dharmendra S Modha, Rajagopal Ananthanarayanan, Steven K Esser, Anthony Ndirango, Anthony J Sherbondy, and Raghavendra Singh. "Cognitive computing". In: *Communications of the ACM* 54.8 (2011), pp. 62–71.

[56] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[57] Sergio Moreschini, Robert Bregovic, and Atanas Gotchev. "Shearlet-Based Light Field Reconstruction of Scenes with Non-Lambertian Properties". In: *2019 8th European Workshop on Visual Information Processing (EUVIP)*. 2019, pp. 140–145. DOI: 10.1109/EUVIP47703.2019.8946124.

[58] Sergio Moreschini, Robert Bregovic, and Atanas Gotchev. "Volumetric segmentation for integral microscopy with Fourier plane recording". In: *IS and T International Symposium on Electronic Imaging: Image Processing: Algorithms and Systems*. 2022.

[59] Sergio Moreschini, Filipe Gama, Robert Bregovic, and Atanas Gotchev. "CIVIT dataset: Horizontal-parallax-only densely-sampled light-fields". English. In: *European Light Field Imaging (ELFI) Workshop*. European Light Field Imaging Workshop ; Conference date: 04-06-2019 Through 06-06-2019. 2019.

[60] Sergio Moreschini, Filipe Gama, Robert Bregovic, and Atanas Gotchev. "CIVIT dataset: Integral microscopy with Fourier plane recording". In: *Data in Brief* 46 (2023), p. 108819. ISSN: 2352-3409. DOI: https://doi.org/10.1016/j.dib.2022.108819. URL: https://www.sciencedirect.com/science/article/pii/S2352340922010228.

[61] Sergio Moreschini, David Hästbacka, and Davide Taibi. "MLOps pipeline development: the OSSARA use case". In: *Proceedings of the Conference on Research in Adaptive and Convergent Systems*. 2023. In press.

[62] Sergio Moreschini, Francesco Lomio, David Hästbacka, and Davide Taibi. "MLOps for evolvable AI intensive software systems". In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2022, pp. 1293–1294. DOI: 10.1109/SANER53432.2022.00155.

[63] Sergio Moreschini, Francesco Lomio, David Hästbacka, and Davide Taibi. "Smart Edge Service Update Scheduler: An Industrial Use Case". In: *Service-Oriented Computing – ICSOC 2022 Workshops*. Springer Nature Switzerland, 2023, pp. 171–182. DOI: https://doi.org/10.1007/978-3-031-26507-5_14.

[64] Sergio Moreschini, Fabiano Pecorelli, Xiaozhou Li, Sonia Naz, Michele Albano, David Hästbacka, and Davide Taibi. "Cognitive Cloud: The Definition". In: *Distributed Computing and Artificial Intelligence, 19th International Conference*. Cham: Springer International Publishing, 2023, pp. 219–229. ISBN: 978-3-031-20859-1.

[65] Sergio Moreschini, Fabiano Pecorelli, Xiaozhou Li, Sonia Naz, David Hästbacka, and Davide Taibi. "Cloud Continuum: The Definition". In: *IEEE Access* 10 (2022), pp. 131876–131886. DOI: https://doi.org/10.1109/ACCESS.2022.3229185.

[66] Sergio Moreschini, Shahrzad Pour, Ivan Lanese, Daniel Balouek-Thomert, Justus Bogner, Xiaozhou Li, Fabiano Pecorelli, Jacopo Soldani, Eddy Truyen, and Davide Taibi. *AI Techniques in the Microservices Life-Cycle: A Survey*. 2023. arXiv: 2305.16092 [cs.SE].

[67] Sergio Moreschini, Gilberto Recupito, Valentina Lenarduzzi, Fabio Palomba, David Hastbacka, and Davide Taibi. *Toward End-to-End MLOps Tools Map: A Preliminary Study based on a Multivocal Literature Review*. 2023. arXiv: 23 04.03254 [cs.SE].

[68] Sergio Moreschini, Gabriele Scrofani, Robert Bregovic, Genaro Saavedra, and Atanas Gotchev. "Continuous Refocusing for Integral Microscopy with Fourier Plane Recording". In: *2018 26th European Signal Processing Conference (EUSIPCO)*. 2018, pp. 216–220. DOI: 10.23919/EUSIPCO.2018.8553138.

[69] Sergio Moreschini, Elham Younesian, David Hästbacka, Michele Albano, Jiří Hošek, and Davide Taibi. *Edge to Cloud Tools: A Multivocal Literature Review*. 2023. arXiv: 2305.17464 [cs.SE].

[70] Håvard Myrbakken and Ricardo Colomo-Palacios. "DevSecOps: a multivocal literature review". In: *Software Process Improvement and Capability Determination: 17th International Conference, SPICE 2017, Palma de Mallorca, Spain, October 4–5, 2017, Proceedings*. Springer. 2017, pp. 17–29.

[71] Sam Newman. *Building microservices*. " O'Reilly Media, Inc.", 2021.

[72] Phu Nguyen, Nicolas Ferry, Gencer Erdogan, Hui Song, Stéphane Lavirotte, Jean-Yves Tigli, and Arnor Solberg. "Advances in deployment and orchestration approaches for IoT-a systematic review". In: *2019 IEEE international congress on Internet of Things (ICIOT)*. IEEE. 2019, pp. 53–60.

[73] Simon Niklaus, Long Mai, and Feng Liu. "Video Frame Interpolation via Adaptive Convolution". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2017, pp. 2270–2279. DOI: 10.1109/CVPR.2017.244. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.244.

[74] Jussi Nupponen and Davide Taibi. "Serverless: What it Is, What to Do and What Not to Do". In: *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*. 2020, pp. 49–50. DOI: 10.1109/ICSA-C50368.2020.00016.

[75] Katie O'Leary and Makoto Uchida. "Common problems with Creating Machine Learning Pipelines from Existing Code". In: 2020.

[76] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. “Cloud container technologies: a state-of-the-art review”. In: *IEEE Transactions on Cloud Computing* 7.3 (2017), pp. 677–692.

[77] Claus Pahl and Brian Lee. “Containers and clusters for edge cloud architectures–a technology review”. In: *2015 3rd international conference on future internet of things and cloud*. IEEE. 2015, pp. 379–386.

[78] Ralf Penners and Andrej Dyck. “Release engineering vs. DevOps-an approach to define both terms”. In: *Full-scale Software Engineering* (2015), p. 49.

[79] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. “Systematic mapping studies in software engineering”. In: *EASE 12*. 2008, pp. 1–10.

[80] Lorena Poenaru-Olaru, Luis Cruz, Arie van Deursen, and Jan S. Rellermeyer. *Are Concept Drift Detectors Reliable Alarming Systems? – A Comparative Study*. 2022. arXiv: 2211.13098 [cs.LG].

[81] Gede Artha Azriadi Prana, Abhishek Sharma, Lwin Khin Shar, Darius Foo, Andrew E Santosa, Asankhaya Sharma, and David Lo. “Out of sight, out of mind? How vulnerable dependencies affect open-source projects”. In: *Empirical Software Engineering* 26 (2021), pp. 1–34.

[82] Saravanan Ramanathan, Nitin Shivaraman, Seima Suryasekaran, Arvind Easwaran, Etienne Borde, and Sebastian Steinhorst. “A survey on time-sensitive resource allocation in the cloud continuum”. In: *it-Information Technology* 62.5-6 (2020), pp. 241–255.

[83] Gilberto Recupito, Fabiano Pecorelli, Gemma Catolino, Sergio Moreschini, Dario Di Nucci, Fabio Palomba, and Damian A. Tamburri. “A Multivocal Literature Review of MLOps Tools and Features”. In: *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2022, pp. 84–91. DOI: 10.1109/SEAA56994.2022.00021.

[84] *Replication Package*. https://github.com/clowee/Exploring-Factors-and-Measures-to-Select-Open-Source-Software. Accessed: 2021-01-07.

[85] Laxmi Rijal, Ricardo Colomo-Palacios, and Mary Sánchez-Gordón. “Aiops: A multivocal literature review”. In: *Artificial Intelligence for Cloud and Edge Computing* (2022), pp. 31–50.

[86] Chaitanya K. Rudrabhatla. “Comparison of zero downtime based deployment techniques in public cloud infrastructure”. In: *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 2020, pp. 1082–1086. DOI: 10.1109/I-SMAC49090.2020.9243605.

[87] Mohammed A Al-Sharafi, Ruzaini Abdullah Arshah, and Emad A Abu-Shanab. “Factors influencing the continuous use of cloud computing services in organization level”. In: *Proceedings of the international conference on advances in image processing*. 2017, pp. 189–194.

[88] Jens Smeds, Kristian Nybom, and Ivan Porres. “DevOps: a definition and perceived adoption impediments”. In: *Agile Processes in Software Engineering and Extreme Programming: 16th International Conference, XP 2015, Helsinki, Finland, May 25-29, 2015, Proceedings 16*. Springer. 2015, pp. 166–177.

[89] Adrian Spataru. “A review of blockchain-enabled fog computing in the cloud continuum context”. In: *Scalable Computing: Practice and Experience* 22.4 (2021), pp. 463–468.

[90] Sergej Svorobej, Malika Bendechache, Frank Griesinger, and Jörg Domaschka. “Orchestration from the Cloud to the Edge”. In: *The Cloud-to-Thing Continuum* (2020), pp. 61–77.

[91] Georgios Symeonidis, Evangelos Nerantzis, Apostolos Kazakis, and George A. Papakostas. “MLOps - Definitions, Tools and Challenges”. In: *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. 2022, pp. 0453–0460. DOI: 10.1109/CCWC54503.2022.9720902.

[92] Davide Taibi. “An Empirical Investigation on the Motivations for the Adoption of Open Source Software”. eng. In: IARIA, 2015, pp. 426–431. ISBN: 978-1-61208-438-1.

[93] Yudai Tanabe, Tomoyuki Aotani, and Hidehiko Masuhara. “A Context-Oriented Programming Approach to Dependency Hell”. In: *Proceedings of the 10th ACM International Workshop on Context-Oriented Programming: Advanced Modularity for Run-Time Composition*. COP ’18. Amsterdam, Netherlands: Association for Computing Machinery, 2018, pp. 8–14. ISBN: 9781450357227. DOI: 10.1145/3242921.3242923. URL: https://doi.org/10.1145/3242921.3242923.

[94] Maruti Techlabs. *9 Key Benefits of DevOps — How to Go About DevOps Transformation Journey*. https://medium.com/geekculture/9-key-benefits-of-devops-54666b8f5c6a. 2021.

[95] *The Cloud Continuum. Be ever–ready for every opportunity*. Accessed: 2023-02-23. URL: https://www.accenture.com/us-en/insights/cloud/cloud-continuum.

[96] Johan L. Thomsen, Kristian D. Schmidt Thomsen, Rasmus B. Schmidt, Søren D. Jakobsgaard, Thor Beregaard, Michele Albano, Sergio Moreschini, and Davide Taibi. "Edge Computing Tasks Orchestration: An Energy-Aware Approach". In: *IEEE International Conference on Edge Computing (EDGE)* (2023). In press.

[97] *Training data for OSSARA Model*. https://doi.org/10.6084/m9.figshare.16944001.v1. Accessed: 2021-11-06.

[98] Mohammad Hadi Valipour, Bavar Amirzafari, Khashayar Niki Maleki, and Negin Daneshpour. "A brief survey of software architecture concepts and service oriented architecture". In: *2009 2nd IEEE International Conference on Computer Science and Information Technology*. 2009, pp. 34–38.

[99] Yingxu Wang. "On cognitive informatics". In: *Brain and Mind* 4.2 (2003), pp. 151–167.

[100] Yi Wei and M Brian Blake. "Service-oriented computing and cloud computing: Challenges and opportunities". In: *IEEE Internet Computing* 14.6 (2010), pp. 72–75.

[101] Brad Wuetherick. "Basics of qualitative research: Techniques and procedures for developing grounded theory". In: *Canadian Journal of University Continuing Education* 36.2 (2010).

[102] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. "All one needs to know about fog computing and related edge computing paradigms: A complete survey". In: *Journal of Systems Architecture* 98 (2019), pp. 289–330. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.02.009. URL: https://www.sciencedirect.com/science/article/pii/S1383762118306349.

[103] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. "All one needs to know about fog computing and related edge computing paradigms: A complete survey". In: *Journal of Systems Architecture* 98 (2019), pp. 289–330.

[104] Liming Zhu, Len Bass, and George Champlin-Scharff. "DevOps and Its Practices". In: *IEEE Software* 33.3 (2016), pp. 32–34. DOI: 10.1109/MS.2016.81.

[105] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. "An Overview of Concept Drift Applications". In: *Big Data Analysis: New Algorithms for a New Society*. Ed. by Nathalie Japkowicz and Jerzy Stefanowski. Cham: Springer International Publishing, 2016, pp. 91–114. ISBN: 978-3-319-26989-4. DOI: 10.1007/978-3-319-26989-4_4. URL: https://doi.org/10.1007/978-3-319-26989-4_4.

APPENDIX A PUBLICATION SUMMARY

Publication I - The work explores the factors considered by companies when selecting OSS for software development and analyzes their availability in OSS portals using APIs. We identified 8 factors, 74 sub-factors, and 170 metrics commonly used to evaluate and select OSS. However, only a small part of the factors can be evaluated automatically, and out of 170 metrics, only 40 are available from project portals APIs. The extraction of information from the 100K most starred GitHub projects revealed that only 22 metrics out of 40 returned information for all the projects. The paper provides an updated list of factors and metrics that practitioners can use to select OSS and can also benefit researchers, OSS producers, and repositories to ease the evaluation of OSS projects.

Publication II - The work presents OSSARA, the OSS Abandonment Risk Assessment model. The model aims at predicting the risk of abandonment for each embedded OSS component and evaluate their criticality in a software system. The prediction is performed based on two factors the probability of each component losing maintenance support during a given timeframe and the significance of each component for the overall system. This enables practitioners to monitor the risk level of the system and decide whether to maintain or replace the OSS components. To validate the model a preliminary assessment was carried out on 12,208 OSS projects that meet the following criteria: contain at least 1,000 commits from at least five unique contributors and are watched by at least 100 users. The four classification algorithms selected were decision tree, support vector machine, logistic regression, and naive Bayes.

Publication III - The work presents a systematic mapping study on Cloud Continuum definitions. From 36 identified definitions we performed an analysis and categorization to propose a new definition. The proposed definition is "an extension of the traditional Cloud towards multiple entities (e.g., Edge, Fog, IoT) that provide analysis, processing, storage, and data generation capabilities." The new definition, based on the concepts of extension of the resources and extension of computational capabilities, can help practitioners and researchers better understand the concept of cloud continuum and advanced service-oriented computing.

Publication IV - The work introduces a smart edge provisioning algorithm that minimizes the number of service drops and maximizes service quality in a service-oriented architecture. The service-oriented system runs on edge nodes and provides services to 270K IoT devices. The algorithm is designed to optimize the environment provided, resulting in a reduced time required to upgrade network components without affecting service availability during peak demand hours. The algorithm has been validated through the use of metrics to compute the performance of the network. The adoption of the algorithm has allowed the company to deploy new updates in a continuous manner while only experiencing a daily drop of 20% of service API calls for a period of 30 minutes.

Publication V - The work discusses the process of creating an MLOps pipeline leveraging open source tools. The proposal has been supported by illustrating a developed use case scenario, specifically the OSSARA model from Publication II has been used. The main aim has been placed on creating a model capable of performing continuous training. By integrating OSS tools such as DugsHub, DVC, and MLflow, it was possible to create a fully automated system that keeps track of different data, models, and experiments when deploying the risk assestment model.

PUBLICATIONS

PUBLICATION I

Exploring factors and metrics to select open source software components for integration: An empirical study

Xiaozhou Li, Sergio Moreschini, Zheying Zhang, and Davide Taibi

Journal of Systems and Software 188 (2022), p. 111255
DOI: https://doi.org/10.1016/j.jss.2022.111255

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Exploring factors and metrics to select open source software components for integration: An empirical study☆

Xiaozhou Li[1], Sergio Moreschini[1], Zheying Zhang, Davide Taibi*

Tampere University, Tampere, Finland

ARTICLE INFO

Article history:
Received 9 January 2021
Received in revised form 6 December 2021
Accepted 1 February 2022
Available online 11 February 2022

Keywords:
Open source
Software selection
Open source adoption

ABSTRACT

Context: Open Source Software (OSS) is nowadays used and integrated in most of the commercial products. However, the selection of OSS projects for integration is not a simple process, mainly due to a of lack of clear selection models and lack of information from the OSS portals.
Objective: We investigate the factors and metrics that practitioners currently consider when selecting OSS. We also investigate the source of information and portals that can be used to assess the factors, as well as the possibility to automatically extract such information with APIs.
Method: We elicited the factors and the metrics adopted to assess and compare OSS performing a survey among 23 experienced developers who often integrate OSS in the software they develop. Moreover, we investigated the APIs of the portals adopted to assess OSS extracting information for the most starred 100K projects in GitHub.
Result: We identified a set consisting of 8 main factors and 74 sub-factors, together with 170 related metrics that companies can use to select OSS to be integrated in their software projects. Unexpectedly, only a small part of the factors can be evaluated automatically, and out of 170 metrics, only 40 are available, of which only 22 returned information for all the 100K projects. Therefore, we recommend project maintainers and project repositories to pay attention to provide information for the project they are hosting, so as to increase the likelihood of being adopted.
Conclusion: OSS selection can be partially automated, by extracting the information needed for the selection from portal APIs. OSS producers can benefit from our results by checking if they are providing all the information commonly required by potential adopters. Developers can benefit from our results, using the list of factors we selected as a checklist during the selection of OSS, or using the APIs we developed to automatically extract the data from OSS projects.

1. Introduction

Open Source Software (OSS) has become mainstream in the software industry, and different OSS projects are now considered as good as closed source ones (Robles et al., 2019; Kilamo et al., 2020). However, selecting a new OSS project requires special attention, and companies are still struggling to understand how to better select them (Lenarduzzi et al., 2020).

One of the main issues during the selection of OSS projects, is the lack of clear information provided by OSS providers about the software quality assessment, and in particular the lack of automated tools that help the selection (Lenarduzzi et al., 2020).

A local company hired our research group to ease and standardize the OSS selection process and to automate it as much as possible, to reduce the subjectivity and the effort needed for the evaluation phase. Currently, the company does not prescribe any selection model, and reported us that their developers commonly struggle to understand what they need to consider when comparing OSS projects.

In this paper, we investigate the first steps towards the definition of a semi-automated OSS evaluation model. Therefore, we extend our previous work (Lenarduzzi et al., 2020) by conducting a survey investigating the factors commonly considered by the companies when selecting OSS, the source of information that can be used to analyze these factors, and the availability of such information on the portals.

The goal of our work is to investigate and determine the factors that practitioners are currently considering when selecting OSS, to identify the sources (portals) that can be used to evaluate such factors mentioned by the practitioners, and to validate the public APIs that can be accessed to automatically evaluate those factors from the sources and portals.

☆ Editor: Raffaela Mirandola.
* Corresponding author.
E-mail addresses: xiaozhou.li@tuni.fi (X. Li), sergio.moreschini@tuni.fi (S. Moreschini), zheying.zhang@tuni.fi (Z. Zhang), davide.taibi@tuni.fi (D. Taibi).
[1] The two authors equally contributed to the paper.

https://doi.org/10.1016/j.jss.2022.111255

The research community has been studying OSS selection and evaluation from different perspectives.

Researchers developed methods to evaluate, compare, and select OSS projects. Such methods and tools exploit different types of approaches, including manual extraction of data from OSS portals (e.g. OMM Petrinja et al., 2009, OpenBQR Taibi et al., 2007a, PSS-PAL Wasserman et al., 2017).

Researchers also proposed platforms for mining data from OSS repositories, that can also be used as the sources of information for the evaluation and comparison of OSS (e.g., The SourceForge Research Data Archive (SRDA) Madey, 2008, FLOSSmole Anon, 2020a, FLOSSMetrics Anon, 2020b, tools to provide dump of existing OSS portals (e.g. GHArchive Anon, 2020c, GitTorrent Anon, 2020d, …), and tools to extract information from OSS portals (e.g. PyDriller Spadini et al., 2018, CVSAnaly Robles et al., 2004...).

Moreover, different approaches to evaluate software quality, often applied to OSS, have been proposed in research, including software metrics (e.g. Chidamber and Kemerer's metrics suite Chidamber and Kemerer, 1994, Cyclomatic Complexity McCabe, 1976), tools to detect technical debt Avgeriou et al., 2020 or to measure other quality aspects (e.g. Software Quality Index Dixon, 2016, Architectural Debt Index Roveda et al., 2018).

Though the previous work provided a significant amount of results on OSS quality evaluation, OSS development data crawling, and OSS selection and adoption models, such works are still limited and not easy to apply in industry for selecting OSS because of various reasons:

- OSS selection models
  - Limited application in industry of the previous OSS selection models Lenarduzzi et al., 2020. The vast majority of models have never been adopted massively by industries with neither case studies nor success stories on the usage of these models therein. One of the potential reasons is that it is nearly not possible to have a generally accepted set of OSS selection criteria to use. The companies must adopt the criteria for their specific needs and constraints to achieve their business objectives. Another reason for such limited adoption of the selection models can be related to the lack of maturity of the models. The models lack clarity and guides about which metrics would offer the most relevant insights into the selection criteria.
- OSS Mining Platforms such as The SourceForge Research Data Archive (SRDA) (Madey, 2008), GHArchive, GitTorrent (Gousios, 2013a)
  - They are designed for research purposes and are complex to use and often have different dependencies for developers that simply need to get data for an OSS project. As an example, GitArchive (Anon, 2020c) does not allow to directly query the data with an API, but needs to be accessed through Google Big Query or dumping the files. Moreover, the collection of all the information needed by the users to evaluate an OSS project requires to use several platforms. This study aims to provide an overview on what information is important to the companies and from what platforms to extract it when evaluating OSS projects.
  - They are often not maintained in the long term. As an example, The SourceForge Research Data Archive (SRDA) (Madey, 2008) is not available anymore. The GHTorrent (Gousios, 2013a) was created in 2013 with the last activity reported in 2019. More information on these platforms is available in Table 1.
- Tools to evaluate software quality
  - They are complex to use and often require effort for manual configuration and analysis on the target software.
  - Most tools require expertise to understand which metrics should be used in which context, and how to interpret the evaluation results. They often provide an overload of information, but not always useful for every context. As an example, tools for assessing the quality and technical debt of software, such as SonarQube,[2] include more than 500 different rules to validate the source code, but only a limited amount of the rules that are commonly associated to specific qualities.
  - When existing tools focus on a specific set of quality metrics to do the evaluation, there is a lack of a tool that can aggregate the factors commonly considered during the selection of OSS.
  - The existence of an OSS project community and of a health ecosystem is an informative indicator of the maturity of a software and its propensity for growth. Even though there are many community-related factors and metrics identified in the OSS selection model, some metrics cannot be accessed directly from the project's repository and existing tools provide very limited support to analyze the data associated with the OSS project community and its support in OSS evaluation.
- Existing Software Quality Models and Metrics such as the Architectural Quality index (Roveda et al., 2018), but also metrics such as the Cyclomatic complexity (McCabe, 1976) or the presence of Code Smells (Fowler, 1999) or anti-patterns (Brown et al., 1998).
  - Are usually targeting mainly on quality, while companies might be interested in other aspects while selecting OSS (e.g. Costs, licenses, features, …).
  - Lack of comparison of the magnitude of the observed effect: different models return different outputs. As an example, previous works indicated that high levels of cyclomatic complexity might result in less readable source code. While besides that, the presence of some smells can be more harmful than others, but the analysis did not take into account the magnitude of such observed phenomenon. As an example, it is not clear if a piece of code with a cyclomatic complexity equals to 10 is twice more complex to read than the same piece of code with that of 5. The same applies to the comparison of several other metrics, including the presence of different amount of code smells. Thus, the comparison of the results of the metrics, increases the complexity of the analysis and comparison between projects.
  - Lack of complex and historical analysis. A complete comparison of an OSS might require not only the analysis of the latest snapshots of a project, but a historical analysis, thus increasing the complexity and the effort required to perform an analysis.

In addition, lack of expertise in companies, in particular on software quality, hinders practitioners to select the most suitable quality models for comparing the projects.

To cope with the aforementioned issues, this paper aims at corroborating and extending previous empirical research on OSS

[2] SonarQube http://www.sonarqube.org.

Table 1
OSS source of information and portals reported in the literature.

Portal	Created on	Last activity	Information stored
OSS aggregators			
(Anon, 2020p) Software Heritage	2018	2020	Source Code
(Anon, 2020f) OpenHub	2004	2020	OSS tracker
(Anon, 2020g) FlossHub	2008	2018	Research portal
(Anon, 2020e) SourceForge Research Data	2005	2008	Statistics
(Anon, 2020c) GH Archive	2012	2020	Timeline Record
(Anon, 2020d) GH Torrent	2013	2019	GH event monitoring
(Anon, 2020a) FLOSSMole	2004	2017	Project data
(Anon, 2020q) PROMISE	2005	2006	Donated SE data
(Anon, 2020b) FLOSSMetrics	2006	2010	Metrics and Benchmarking
Audit and analysis tools			
(Anon, 2020k) WhiteSource	2011	2020	Security
(Anon, 2020r) FossID	2016	2020	OS Compliance and Security
(Anon, 2020j) Synopsys (formerly BlackDuck)	2012	2020	legal, security, and quality risks
(Anon, 2020l) SonarQube	2006	2020	Code quality and security
(Anon, 2020n) WhiteHat	2001	2020	Software composition analysis
(Anon, 2020l) SonarCloud	2008	2020	Software quality analysis
OSS mining data tools			
(Anon, 2020s) BOA	2015	2019	Source code mining
(Anon, 2020h) Candoia	2016	2017	Software repository mining
(Anon, 2020i) RepoGrams	2016	2020	OSS Comparison
Questions and answers portal			
(Anon, 2020t) Stack Exchange	2009	2020	Q&A
(Anon, 2020u) Reddit	2009	2020	Q&A

selection and adoption, so as to enable, not only our target company to assess and compare OSS but also other companies. More specifically, this study aims at extending our previous work (Lenarduzzi et al., 2020) from different point of views:

- We performed a survey to update the common factors that are currently considering when selecting OSS and we compared them with the factors considered in the past (elicited in the Euromicro/SEAA SLR Lenarduzzi et al., 2020)
- We analyzed the source of information and portals that can be used to assess the aforementioned factors
- We analyzed the public APIs that can be accessed to automatically assess the factors from the aforementioned portals
- We extracted the information for 100K projects, to validate their availability.

The source of information associated with the factors and metrics adopted to measure them will help developers to understand and adopt OSS selection models for their specific needs and constraints. Moreover, they will help to remind OSS producers to provide information commonly expected by the potential users of the software.

Together with the validated APIs to automatically extract the assessment information, the result of the work forms a critical step towards developing semi-automatic tools to facilitate the practice of OSS selection.

The remainder of this paper is structured as follows. Section 2 presents related works. Section 3 describes the research method we adopted to achieve our goals. Section 4 reports the results while Section 5 discuss them. Section 6 finally draws conclusions and future works.

2. Related work

To cope with the need of selecting valuable OSS projects, several evaluation models have been proposed (e.g. Duijnhouwer and Widdows (2003), Golden (2008), Taibi et al. (2007b) and Semeteys (2008)). At the same time, different research groups proposed project aggregators to ease the access of different information on OSS, measures and other information. Last, but not least, research in mining software repositories also evolved in parallel, and different researchers provided datasets of OSS projects, portals and tools to extract information from OSS projects.

In the remainder of this Section, we summarize related work on the factors adopted to evaluate OSS, OSS evaluation and selection models, OSS aggregator portals, tools for OSS repository mining and tools for OSS analysis and audit.

2.1. The factors considered during the adoption of OSS

In the systematic literature review (SLR) on OSS selection and adoption models (Lenarduzzi et al., 2020) that we are extending in this work, we analyzed 60 empirical studies, including 20 surveys, 5 lessons learned on OSS adoption motivation and 35 OSS evaluation models.

Regarding the common factors of OSS selection and adoption, eight main categories were reported by the selected studies, including, *Community and Adoption*, *Development process*, *Economic*, *Functionality*, *License*, *Operational software characteristics*, *Quality*, *Support and Service*. For each category, sub-factors or metrics are reported. Results show that not all factors were considered equally important according to evaluation models and to surveys and lessons learned. For example, factor *cost* is considered much more important by the surveys than by the models when, on the contrary, the importance of factor *maturity* is seen oppositely. Furthermore, certain factors are considered important by both groups, such as, *Support and Service*, *Code Quality*, *Reliability*, etc.

Table 1 lists sources of information mentioned in the related works to assess the common factors considered during the adoption of OSS. The table only reports the indirect sources of information. Direct sources of information such as the official portal, or the versioning system (e.g. GitHub, GitLab) are not mentioned in the table.

2.2. OSS evaluation and selection models

Within the 35 OSS models identified in our previous literature review (Lenarduzzi et al., 2020), 21 (60%) were built via case study, with 5 via interview, 5 via experience and the other 4 via the combination of interview and case study. All proposed models provide either checklist (13 models) or measurement (8 models) or both (14 models) as their working approaches. On the other hand, regarding the studies with surveys and lesson learned, the majority (13) target at adoption motivation identification with the remainders on other scopes. Furthermore, 12 tools are introduced in 22 of the given studies; however, only two out of the 12 are properly maintained.

All the models propose to evaluate OSS with a similar approach:

- *Identification of OSS candidates*. In this step, companies need to identify a set of possible candidates based on their needs.
- *Factors evaluation* A list of factors are then assessed, by extracting the information or measures from the OSS portals, or by measuring/running the project candidates
- *Project scoring* The final score is then normalized based on the importance of each factor and the final evaluation is computed.

The Open Source Maturity Model (OSMM), was the first model proposed (Duijnhouwer and Widdows, 2003; Golden, 2008) in the literature. OSMM is an open standard that aims at facilitating the evaluation and adoption of OSS. The evaluation is based on the assumption that the overall quality of the software is proportional to its maturity. The evaluation is performed in three steps:

1. Evaluation of the maturity of each aspect. The considered aspects are: the software product, the documentation, the support and training provided, the integration, the availability of professional services.
2. Every aspect is weighted for importance. The default is: 4 for software, 2 for the documentation, 1 for the other factors.
3. The overall maturity index is computed as the weighted sum of the aspects' maturity.

The OSMM has the advantage of being simple. It allows fast (subjective) evaluations. However, the simplicity of the approach is also a limit: several potentially important characteristics of the products are not considered. For instance, one could be interested in the availability of professional services and training, in details of the license, etc. All these factors have to be 'squeezed' into the five aspects defined in the model.

The Open Business Readiness Rating (OpenBRR) (Wasserman et al., 2006) is an OSS evaluation method aiming at providing software professionals with an index applicable to all the current OSS development initiatives, reflecting the points of view of large organizations, SMEs, universities, private users, etc. The OpenBRR is a relevant step forward with respect to the OSMM, since it includes more indicators, the idea of the target usage, and the possibility to customize evaluations performed by other, just by providing personalized weights. With respect to the latter characteristics, the OpenBRR has however some limits: one is that for many products it is difficult to choose a "reference application" that reflects the needs of the users; another is that there are lots of possible target usages, each with its own requirements; finally, the evaluation performed by a user could be not applicable to other users. In any case, the final score is a synthetic indicator to represent the complex set of qualities of a software product. On the official OpenBRR site several evaluations were available, and originally provided as spreadsheet. However the OpenBRR website and tools are not available anymore.

The Qualification and Selection of Open Source Software (QSOS) (Semeteys, 2008) works similarly as OpenBRR, but requires first to create an Identity Card (IC) of each project, reporting general information (name of the product, release date, type of application, description, type of license, project URL, compatible OS, …), then to evaluate the available services, functional and technical specifications and grade them (in the 0..2 range). Then, evaluators can specify the importance of the criteria and their constraints. Finally a normalized score is computed to compare the selected project candidates. Although the method is effectively applicable to most OSS, the QSOS approach does not represent a relevant step forward with respect to other evaluation methods. Its main contribution is the set of characteristics explicitly stated which compose the IC, and the provision of a guideline for the consistent evaluation of these characteristics. The evaluation procedure is rigid. For instance, it requires to define the IC of each OSS under evaluation, even if they are not completely matching the requirements. Such a procedure is justified when the ICs of products are available from the OS community before a user begins the evaluation. However even in this case it may happen that the user needs to consider aspects not included in the IC: this greatly decreases the utility of ready-to-use ICs. The strict guidelines for the evaluation of the IC, necessary to make other users' scoring reusable, can be ill suited for a specific product or user. Finally, even though in the selection criteria it is possible to classify requirements as needed or optional, there is no proper weighting of features with respect to the intended usage of the software.

OpenBQR (Taibi et al., 2007a) works in a similar way as OpenBRR, but requires the evaluators to first specify the importance of the factors, and then to assess the projects, so as to avoid to invest time evaluating factors that are not relevant for the specific context. OpenBQR is an important step forward in terms of effort required to evaluate the projects.

OSS-PAL (Wasserman et al., 2017) works similarly as QSOS, but proposed to introduce a semi-automated evaluation, supported by an online portal. Unfortunately, the portal seems to be only a research prototype, and does not collect any data automatically.

All the aforementioned models have some drawbacks:

- Existing methods usually focus on specific aspects of OSS. For example, the OSMM focuses on software maturity, but misses some potentially interesting characteristics like license compliance or security for the quality assessment. On the other hand, methods like OpenBRR, QSOS, OpenBQR, etc. provide a set of indicators reflecting a wide range of potential users' viewpoints for the quality assessment. This requires individuals to identify the importance of assessment factors according to their needs and introduces extra effort and complexity to adopt a method for practice.
- The OSS evaluation requires effort to run the software and to extract information from the OSS portals. The assessment process of existing methods is not optimized. Methods such as QSOS proceed to evaluate indicators before they are weighted, so some factors may be measured or assessed even if they are later given a very low weight or even a null one. This results in unnecessary waste of time and effort.
- The dependence of the users of OSS is not adequately assessed, especially the availability of support over time and the cost of proprietary modules developed by third parties.

2.3. OSS aggregators

Many platforms have been developed to collect and share OSS-related data, enabling a quick extraction of the information on different OSS projects.

Ohloh was one of the first project aggregators (Bruntink, 2014; Allen et al., 2009) on the market (2004) aimed at indexing several projects from different platforms (GitHub, SurceForge, …). In 2009, Ohloh was acquired by Geeknet, owners of SourceForge (Anon, 2020e) that then sold it to Black Duck Software in

2010. Black Duck, was already developing a product for OSS audit, with a particular focus on the analysis of the license compatibility, and integrated Ohloh's functionality with their products. In 2014, Ohloh became "Black Duck Open Hub" (Anon, 2020f). Finally, Synopsys acquired Black Duck and renamed the Black Duck Open Hub into "Synopsys Open Hub". Synopsis Open Hub is currently the only continuously updated OSS aggregator that include information of different OSS projects from different sources (versioning control systems, issue tracking systems, vulnerability databases). On January 2021, OpenHub indexed nearly 500K projects, and more than 30 billions of lines of code. It provides flexibility for users to select the metrics to compare project statistics, languages, repositories, etc. However, it lacks the OSS evaluation facilities that allow to adjust the importance of selected metrics according to users' needs for automatically scoring the candidate software. In addition, it lacks information related to the community popularity, documentation, availability of questions and answers and other information.

Other OSS aggregators have been proposed so far. FlossHub (Anon, 2020g) and FLOSSMole (Anon, 2020a) had similar goal of OpenHub. However, they have not been updated in the last years. FlossMetrics (Anon, 2020b) had the goal of providing software metrics on a set of OSS. However, it has also been abandoned in 2010.

The Software Heritage (Di Cosmo and Zacchiroli, 2017), differently than the previously mentioned platforms, has the goal of collecting and preserving the history of software projects, and is not meant to enable the comparison or to provide support for selecting OSS. The project is sponsored by different companies and foundations, including the UNESCO foundation. The Software Heritage could be used as a source of information to analyze the activity of a project. However, its access is not immediate, and users need to use APIs to get detailed data on the projects.

Other platforms, designed for supporting mining software activities, might also be used for obtaining relevant information from OSS. In particular, the Sourceforge research data archives (Madey, 2008) shared the SourceForge.net data with academic researchers studying the OSS software phenomenon; GH Archive (Anon, 2020c) records the public GitHub timeline and makes it accessible for further analysis; and the GHTorrent project (Gousios, 2013b) creates a mirror of data offered through the Github REST API to monitor the event timeline, the event contents, and the dependencies.

2.4. OSS repository mining tools

Besides the platforms that aggregate heterogeneous metric providers to track repositories associated with a wide range of OSS projects, there are also research prototypes or projects to mine information from given repositories. In particular, BOA (Dyer et al., 2013, 2015) provide support to mine source code and development history from project repositories using the domain-specific language. Candoia (Anon, 2020h) also provided a platform for mining and sharing information from OSS projects. RepoGrams (Rozenberg et al., 2016; Anon, 2020i) allows to visually compare projects based on the history of the activity of their git repositories.

Other groups developed tools not aimed at supporting the selection of OSS, but that can be used as valuable sources of information. As an example, PyDriller (Spadini et al., 2018) can be used to obtain detailed information from commits.

Surprisingly, none of the previously mentioned papers cited other tools such as Cauldron or SourceCred. Cauldron[3] is a free open source software that is used to collect information from multiple sources as different information are retrieved. SourceCred[4] is an OSS technology which analyzes a project and determines the contributions of individuals in it. It is built on the idea that communities matters but also that the work of singles need to be visible and rewardables.

The Community Health Analytics Open Source Software (CHAOSS)[5] project. CHAOSS, a Linux Foundation project, also developed tools to measure OSS projects, and in particular to measure community health, to analyze software community development and to develop programs for the deployment of metrics not attainable through online trace data.

Different European projects also developed tools for mining data from OSS repositories. The EU H2020 CROSSMINER project[6] (Rocco et al., 2021) includes techniques and tools for extracting knowledge from existing open source components generating relevant recommendations for the development of user's projects. The recommendation system focuses on 4 main activities:

- **Data Preprocessing:** containing tools to extract metadata from repositories
- **Capturing Context:** uses metadata to generate knowledge for mining functionalities
- **Producing Recommendation:** IDE to generate recommendations.
- **Presenting Recommendation:** IDE to show recommendations.

QUALOSS (Quality in Open Source Software) (Soto and Ciolkowski, 2009) and QualiSPo (Quality Platform for Open Source Software) (Del Bianco et al., 2010) projects aimed at identifying quality models to evaluate the quality and the trustworthiness of OSS. Both projects proposed different tools for extracting data from repositories, to calculate software metrics and to identify possible issues in the code or in the community activity. However, none of the tools developed is currently active, and several of them are not available anymore (e.g., QualiSPo Del Bianco et al., 2010)

2.5. OSS audit and analysis tools

Companies like Synopsys (formerly BlackDuck) (Anon, 2020j) and WhiteSource (Anon, 2020k) provide solutions to software composition analysis and offer services of the assessment of OSS quality and code security. Synopsis focuses on their professional services of the license compatibility while WhiteSource emphasizes the open source management to offer services such as viewing the state of OSS components, their license compliance, and the dependencies; prioritizing components' vulnerabilities based on how the proprietary code is utilizing them; analyzing the impact of the vulnerabilities, etc.

Different tools to assess specific qualities are also available on the market. As an example, companies can use tools such as SonarQube (Anon, 2020l) or Sonatype (Anon, 2020m) to evaluate different code-related qualities such as the standard compliance or the technical debt. Or Security-specific tools such a White-Hat Security (Anon, 2020n), Kiuwan (Anon, 2020o) or others to evaluate the security vulnerabilities.

[3] **Cauldron:** https://cauldron.io.

[4] **SourceCred:** https://sourcecred.io/docs/.

[5] **CHAOSS project:** https://chaoss.community/about/.

[6] **CROSSMINER project:** https://www.crossminer.org.

2.6. Gaps of the current OSS assessment models

The different OSS assessment models, tools and platforms provide a possibility to assess OSS projects mainly from the perspectives of license obligation, application security, code quality, etc. They are about the state of software and its quality and comprise an essential part of the assessment model for OSS selection and adoption (Sbai et al., 2018; Lenarduzzi et al., 2020). Besides, activities, supports, or other projects surrounding a project form an important perspective demonstrating if a project exists in a lively ecosystem (Jansen, 2014). In particular, metrics such as response times in Q&A forums and bug trackers, the active contributors and their satisfaction, the user's usage and their satisfaction, the number of downloads, the number of forks, bug-fix time, etc. are informative references indicating the productivity and a propensity for growth of the OSS project community. Some of the measures can be cross-referenced from different data sources, while some need further analysis based on the collected data. To the best of our knowledge, no portal has effectively taken these community-related factors into account when providing service to evaluate and compare OSS projects.

Furthermore, companies have their distinct strategies, needs, and constraints to adopt OSS projects in software development (Lenarduzzi et al., 2020). After practitioners identify a list of candidates that cover the expected features, meet requirements, and fit with the existing technical solution, they specify the importance of the selection criteria , complying with the company's needs and restrictions. As highlighted in our previous systematic literature review (Lenarduzzi et al., 2020), it is impractical for companies to study every software assessment model to select the one that fits their needs best. Therefore, there is a need to call for the OSS evaluation and selection tools that not only guides developers to adapt OSS assessment criteria by identifying and weighing the ones fitting in a specific scenario, but also automates the process of assessing and comparing among a set of selected software based on information which can be extracted from the public APIs of available portals.

3. Research method

3.1. Goal and research questions

Our goal is to investigate and determine the factors that practitioners are currently considering when selecting OSS, to identify the sources (portals) that can be used to evaluate such factors mentioned by the practitioners, and to validate the public APIs that can be accessed to automatically evaluate those factors from the sources and portals.

To achieve the aforementioned goals, we defined four main research questions(RQs).

RQ1. What factors are practitioners considering when selecting OSS projects to be integrated in the software they develop?
In this RQ we aim at collecting the information adopted by practitioners when selecting projects to be integrated in the software they develop. We are not considering OSS products supporting software development process and the management such as IDEs, Office Suites, but software libraries, frameworks or any other tool that will be integrated and packaged as part of the product developed by the company.

RQ2 Which metrics are used by practitioners to evaluate the factors adopted during the selection of OSS?
In this RQ we aim at identifying the metrics adopted by practitioners to evaluate the factors they are interested to assess. As an example, practitioners might assess the size of the community checking the number of committer in the repository, or might check the size of the project by checking the number of commits in the repository or even downloading the software and measuring its size in lines of code.

RQ3 Which source of information and portals are used to assess OSS?
In this RQ we aim at understanding which portals or other sources are used by practitioners to evaluate the factors identified in RQ1, based on the metrics reported in RQ2.

RQ4 Which factor can be extracted automatically from OSS portals?
In this RQ, we aim to systematically analyze the common portals hosting OSS, to identify the information that can be extracted via APIs.

In order to answer our RQs, we conducted our work in three main steps:

Step 1: Interviews among experienced software developers and project managers to elicit the factors affecting the OSS selection (RQ1), the metrics (RQ2) and the sources of information they adopt (RQ3).
Step 2: Analysis of the APIs of the source of information (portals) identified in RQ3.
Step 3: Analysis of the availability of the metrics collected in the previous step (RQ2) in the public API of the sources of information adopted by practitioners (RQ3) among 100k projects (RQ4).

Fig. 1 depicts the process adopted in this work. The detailed process is reported in the remainder of this section.

3.2. Step 1: Interviews on the factors considered when selecting OSS

In order to elicit the factors adopted by practitioners when selecting an OSS in the software product development process, we designed and conducted a semi-structured interview based on a questionnaire.

3.2.1. The interview population

We identified the population for our interviews considering participants who can best provide the information needed in order to answer our RQs. We selected participants that fulfilled the following criteria:

- Currently developing software projects. With this criteria, we aim at selecting participants that are still working on software projects. This criteria will exclude persons that had a long experience but are not working anymore in software development projects (e.g. upper managers)
- At least 5 years of experience in developing software projects. We aim at including only practitioners with a minimum level of experience, excluding freshman and newly graduated ones.
- At least 3 years of experience in the domain they are working. We want to consider only practitioners that have a minimum level of experience in the domain they are working, to avoid incongruences due to the lack of knowledge of the domain.
- At least 3 years of experience in deciding which OSS component integrate in the product they develop.

3.2.2. The questionnaire

The interviews were based on the same questionnaire adopted in our previous works to elicit the factors considered important for evaluating OSS (Del Bianco et al., 2009; Taibi, 2015). We organized the questions in the questionnaire adopted for the interviews two sections, according to the types of information we sought to collect:

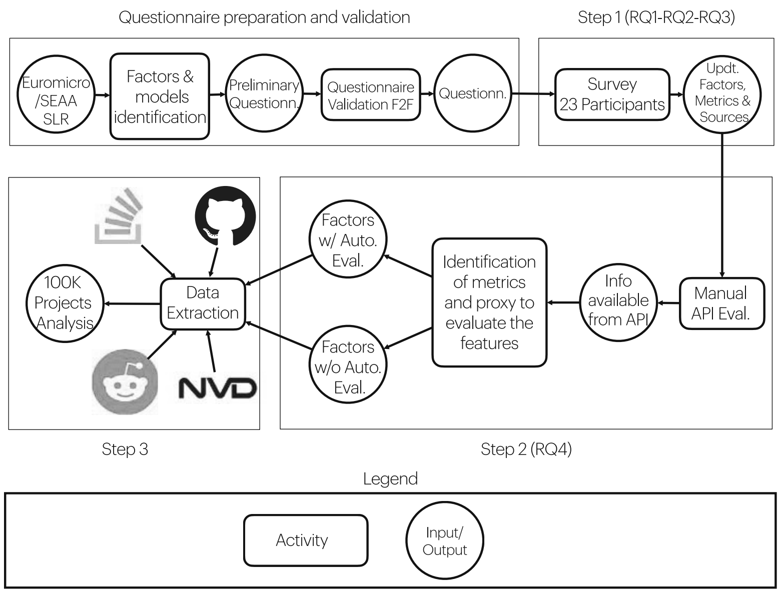


Fig. 1. The study process.

- ***Demographic information***: In order to define the respondents' profile, we collected demographic background information in relation to OSS, including the number of years of experience in selecting OSS components to be integrated in the software they develop. This information considered predominant roles and relative experience. We also collected company information such as application domain, organization's size via number of employees, and number of employees in the respondents' own team.
- ***Factors considered during the adoption of OSS***: Here we asked to list and rank the factors considered during the adoption of OSS software to be integrated in the products they develop, based on their importance, on a 0-to-5 scale, where 0 meant "totally irrelevant" and 5 meant "fundamental".
  - We first asked to list the factors the respondents consider when adopting an OSS in the software product they develop, and to rank the them on the 0-to-5 scale. This open question is to encourage respondents to identify the important factors which might not be clarified in our Euromicro/SEAA SLR (Lenarduzzi et al., 2020).
  - Then, we asked to rank other possible factors not mentioned in the previous step, on the 0-to −5 scale. Please note that the interviewer listed the remaining factors identified in our Euromicro/SEAA SLR and not mentioned by the participant (Lenarduzzi et al., 2020). The factors identified in the Euromicro/SEAA SLR are reported below.
    - Community & Support
    - Documentation
    - Economic
    - License
    - Operational SW Characteristics
    - Maturity
    - Quality
    - Risk
    - Trustworthiness
  - For each factor ranked higher or equal than 3, we asked to:
    - Report the related sub-factors and their associated metrics with the importance ranking on the 0-to −5 scale
    - Report the source they commonly use to evaluate them (e.g. GitHub, Jira, manual inspection, ...)
    - Report the metrics they adopt to measure the factor
  - We finally asked if they think the factors they reported enable a reasoned selection of OSS or if they would still need some piece of information to have a complete picture of the assessment.

The complete questionnaire adopted in the interviews is reported in the replication package (Anon, 2021).

3.2.3. Interviews execution

The interviews were conducted online, using different videoconferencing tools (Zoom, Skype and Microsoft Teams), based on the tool preferred by the interviewed participant. Interviews were carried out from September 2020 to December 2020.

Because of time constraint, and of the impossibility to conduct face-to-face interviews during public events, interviewees were selected using a convenience sampling approach (also known as Haphazard Sampling or Accidental Sampling) (Battaglia, 2008). However, we tried to maximize the diversity of the interviewees, inviting an equal number of developers from large and medium companies, and from companies in different domains. The selected participants are experienced developers or project managers, and have been involved in the OSS selection process or the software integration and configuration management process. We did not consider any profiles coming from academia, such as researchers or students, nor any inexperienced or junior profiles.

3.2.4. Interviews data analysis

Nominal data on the factor importance is determined by the proportion of responses in the according category. In order to avoid bias, the interviewees are asked to recall the important factors without being provided with options. Thus, the proportion of interviewees who mentioning a factor shall reflect its importance fairly. Ordinal data, such as 5-point Likert scales, was not converted into numerical equivalents to prevent the risk of misleading to subsequent analysis. Apparently when the deviation of the responses is large, such a phenomenon will be overlooked. In this way, we can better identify the potential distribution of the interviewees' responses.

Open questions (application domain, other factors reported, platforms adopted to extract the information and metrics adopted to evaluate the factors) were analyzed via open and selective coding (Wuetherick, 2010). The answers were interpreted by extracting concrete sets of similar answers and grouping them based on their perceived similarity. Two authors manually provided a hierarchical set of codes from all the transcribed answers, applying the open coding methodology (Wuetherick, 2010). The authors discussed and resolved coding discrepancies and then applied the axial coding methodology (Wuetherick, 2010).

3.3. Step 2: Analysis of the APIs of the OSS portals

We manually analyzed the APIs of the portals identified in RQ3, looking for APIs that allowed to assess the information needed to measure the factors reported by the interviewees (RQ1 and RQ2). The first two authors independently analyzed all the portals seeking for these pieces of information, and then compared the results obtained. In case of discrepancies, all the incongruities were discussed by all the authors, reaching a 100% consensus.

Some factors were not directly analyzable. For example, the responsiveness of an OSS community cannot be directly measured; hence, a proxy metric, i.e., the average time spans between the created time of issues and the first actions, is adopted. Therefore, the first two authors proposed a list of proxy metrics, considering both the metrics adopted by the interviewees and metrics available in the literature. Then, all metrics were discussed by all the authors until we reach a consensus.

However, as expected, not all the metrics can be automatically extracted, and some of them require a manual assessment. An example of a factor that cannot be automatically extracted is the availability of complete and updated architectural documentation.

3.4. Step 3: Analysis of projects that provide information to assess the factors

3.4.1. Validation of the factors analyzability on the OSS portals

This step was based on three sub-steps:

- *Project selection*. We selected the top 100K GitHub projects, based on the number of stars. The list of selected projects were determined on 2020-11-10. The number of projects was limited to the time available. In particular, the different APIs limit the number of queries that can be executed in one hour, and therefore we limited the study to 100k most starred projects to ensure that the data can be extracted in 2 months. We are aware that some projects might not be code-based projects, and some repositories might only have the purpose to collect resources. However, since it is not possible to automatically exclude non-code projects, we consider them all.
- *Information extraction*. We extracted the selected information from the APIs. We decided to extract only the information needed to evaluate the factors. Other information are available, but requires to run a higher number of queries, and therefore would have reduced the number of projects that we can extract. The extraction process started on 2020-11-16 with data collected gradually till 2020-12-29. As an example, it would be possible to extract all the details on project issues (issue title, author, date, comments, ...), but this would have required to run a number of additional queries, without providing any information considered valuable by our interviewees.
- *Analysis of the information available*. In this step we analyzed which information is actually available for each project. As an example, not all the projects might use different issue trackers instead of using the one provided by GitHub, or some projects might not be listed by the NIST NVD database, or more, some project might not have questions and answers available in StackOverflow or Reddit.

3.5. Replication

In order to ease the replication of this work we provide the complete replication package including the questionnaire adopted for the interviews, the results obtained in the interviews, the data crawling script and the results of the data analysis (Anon, 2021).

4. Results

Here we first provide information about the sample of respondents, which can be used to better interpret the results and then, we show the collected results with a concise analysis of the responses obtained, with insights gained by statistical analysis.

We collected 23 interviews from experienced practitioners. Fig. 2 contains the distribution of company sizes where our interviewees belong, Fig. 3 shows the percentage for organizational roles identified in the questionnaire while Fig. 4 shows the distribution of the experience of our interviewees in selecting OSS components to be integrated in the projects they develop.

4.1. RQ1: Factors considered by practitioners when selecting OSS

Our interviewees consider 8 main factors and 46 sub-factors when they select OSS, reporting an average of 2.35 factors per interviewee, a minimum of 1 and a maximum of 21 factors.

The factor that is mentioned more frequently from the interviewees is *License* which has received a median importance of 4 out of 5. Surprisingly, this is not the value which has received the highest median value of importance as *Community Support and Adoption, Performances* and *Perceived Risk* received a median value of importance of 4.5 out of 5. It is interesting to note that no participant mentioned *economic* and its related sub-factors such as license costs, or cost for training. So this factor is not reported in our results.

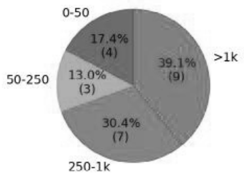
In Table 2, we report the list of factors and sub-factors together with the number of participants who mentioned them (column RQ1- #) and the median of the importance reported by the interviewees (column RQ1 - Median).

4.2. RQ2: Metrics used by practitioners to evaluate factors during OSS selection

When we asked practitioners to report the metrics they use to evaluate the factors they mentioned in RQ1, and to rank their usefulness, practitioners mentioned 110 different metrics.

Company Size (# Employees)	# Interviewees
0-50	4
50-250	3
250-1000	7
>1000	9

(a)

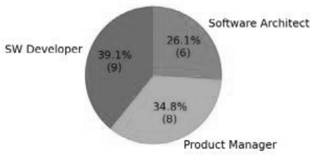


(b) Distribution of company size

Fig. 2. Distribution of company sizes of our interviewees.

Roles	# Interviewees
SW developer	9
Product Manager	8
Software Architect	6

(a)



(b) The roles of our interviewees

Fig. 3. Distribution of roles of our interviewees.

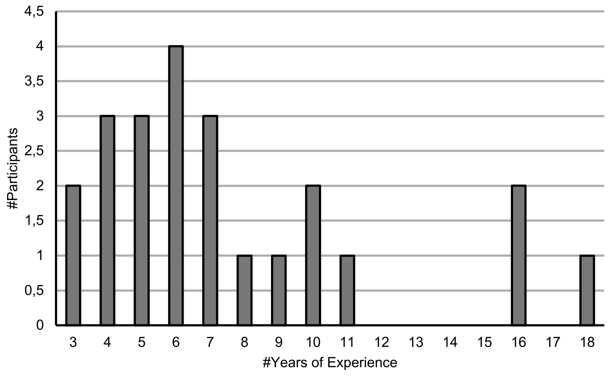


Fig. 4. Number of years of experience of the participants in selecting OSS components to be integrated in the products they develop.

The complete list of metrics reported for each factor is reported in Appendix.

In Table 2 (Column RQ2 - #Metrics) we report the count of metrics considered as useful by practitioners (likert scale $\geq$ 3, where 0 means "This metric is useless to evaluate this factor" and 5 means that the metric is extremely useful).

Surprisingly, the factor where practitioners provided the highest number of metrics to assess it, is *Maturity*, which has been mentioned only 6 times compared to the *License*, mentioned 21 times, where practitioners provided 7 metrics instead. This indicates a wide variety of interpretations on *Maturity*, and practitioners use different metrics to evaluate this factor. The careful reader can also observe that for some factors considered as relevant in RQ1 such as *Perceived risks*, no metrics have been mentioned. This result proves that in some cases, some of the most important factors in an OSS cannot be objectively measured and the interviewees do not know how to retrieve such information appropriately.

4.3. RQ3: Sources of information and portals used to assess OSS

Our interviewees mentioned 9 different source of information and portals they commonly consider when they select OSS.

In Table 3 we list the sources of information adopted by the practitioners to evaluate OSS, together with the number of participants who mentioned it (columns # and % of mentions). To increase the readability, we grouped the source of information in five main categories: *version control systems*, *issue tracking systems*, *Question and Answer portals* (Q&A), *forum and blogs* and *security* related platforms.

The 5 most reported sources of information are GitHub (and GitHub Issue tracker), StackOverflow, Reddit, and NIST Security Vulnerability (NVD) with respectively: 23, 19, 12, 12 and 14 mentions. All of these are mentioned from more than 20% of the interviewees and are therefore those which prove to be the most useful when retrieving information related to OSS. Other platforms such as Bitbucket or Jira were rarely mentioned. The results presented in Table 3 could also be useful to other OSS stakeholders such as software administrators and software operators.

4.4. RQ4: Factors that can be extracted automatically from the portals

We focused our attention to the four most used sources of information reported in RQ3: GitHub (Anon, 2020v), StackOverflow (Anon, 2020w), Reddit (Anon, 2020u) and the NIST National

Table 2
High-level Factors considered during the adoption of OSS.

RQ1			RQ2
Factor	#	Median	#Metrics
Community support and adoption	10	4.5	
Popularity	9	3	4
Community reputation	11	3	3
Community size	13	3	5
Communication	6	3.5	5
Involvement	9	3	1
Sustainability	11	3	1
Product Team	5	3	2
Responsiveness	1	5	1
Documentation	14	4	
Usage documentation	4	4	5
Software requirements	11	3	1
Hardware requirements	8	3.5	1
Software Quality Documentation	5	3	3
License	21	4	7
Operational SW characteristics	6	4	
Trialability	5	3	2
Independence from other SW	11	3	4
Development language	5	4	3
Portability	1	4	1
Standard compliance	5	4	0
Testability	6	3.5	0
Maturity	6	3.5	11
Quality	6	3.5	
Reliability	3	4	6
Performances	4	4.5	1
Security	15	4	6
Modularity	3	3	1
Portability	3	4	2
Flexibility/Exploitability	3	3	3
Code Quality	13	4	6
Coding conventions	9	3	0
Maintainability	3	4	0
Testability	2	4	0
Existence of benchmark/test	4	3.5	4
Changeability	2	3.5	0
Update/Upgrade/Add-ons/Plugin	3	4	1
Architectural quality	5	3	0
Risk (Perceived risks)	7	4.5	
Perceived lack of confidentiality	5	1	0
Perceived lack of integrity	5	3	0
Perceived high availability	5	4	3
Perceived high structural assurance	5	2	0
Strategic risks	5	3	0
Operational risks	5	1	1
Financial risks	5	2	0
Hazard risks	5	4	5
Trustworthiness	6	4	
Component	4	3.5	3
Architecture	4	3	2
System	4	3.5	3
OSS provider reputation	4	3.5	0
Collaboration with other product	4	2.5	3
Assessment results from 3rd parties	2	3.75	0

Table 3
The source of information reported by the interviewees (RQ3).

ID	Source of information	#	% of mentions
	Version control systems:		
R	GitHub Anon (2020v)	23	100
R	GitLab Battaglia (2008)	1	4.3
R	SourceForge Wuetherick (2010)	1	4.3
R	Bitbucket Anon (2020v)	1	4.3
	Issue tracking systems:		
I	GitHub Issues Anon (2020v)	19	82.6
I	Jira Anon (2020w)	1	4.3
	Question and answer portals:		
Q	StackOverflow Anon (2020w)	12	51.2
Q	Reddit Anon (2020u)	12	51.2
	Forum and blogs:		
F	Medium Cai and Zhu (2016)	5	5
F	Hackernews Hu et al. (2012)	5	5
	Security:		
S	CVE Anon (2020y)	1	4.3
S	CVSS Anon (2020y)	2	8.7
S	CWE Anon (2020z)	1	4.3
S	NVD Anon (2020x)	14	21.7

Vulnerability Database (Anon, 2020x). For such purpose, we first identified the APIs that can be adopted to extract the information needed to measure the factors, and then we extracted the data from the 100k with more stars in GitHub.

The extraction of the information for 100K projects took a total of 5 days for GitHub, 53 days for StackOverflow, 4 days for Reddit and 2 Days for the NIST National Vulnerability Database. The long processing time is due to the limit of queries that can be performed on the APIs for different IP addresses.

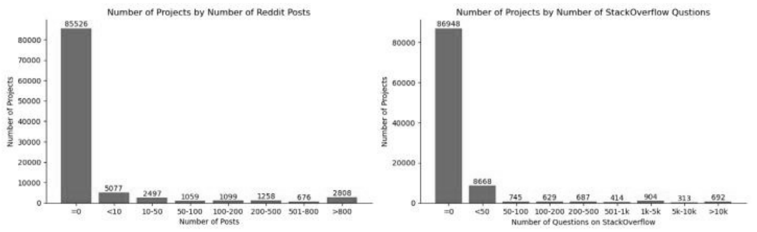
Considering the projects extracted (Fig. 6(a)), more than half of these projects (53.3%) have been active for 2–6 years. Around 12.1% of these projects are active for one year or less when only 3.9% of them are active for more than 10 years. Majority of these projects (87.0%) have less than 500 issues during their life cycle when around 3.4% of them have more than 1k issues (Fig. 6(b)). Furthermore, there are 1791 projects being very popular having more than 10k stars when 25.9% projects having stars ranging from 1k to 10k (Fig. 6(c)) with 46.7% having less than 500 stars. On the other hand, regarding project size, more than half of them (52.8%) have lines of code (LOC) ranging from 20k to 500k. 6.1% projects contain more than 5 m LOC when only 0.9% of them have less than 1k LOC (Fig. 6(d)). Regarding developing languages, Javascript is the most popular being the primary language of OSS projects (17k projects) with Python and Java at 2nd and 3rd (Fig. 6(e)). They are the primary languages for 40.9% of the projects. However, regarding LOC by languages, C language (57.6b) and Javascript (57.2b) rank at the top. Both have almost doubled the amount of C++ language (31.7b) which ranks the 3rd in terms of total LOC (Fig. 6(f)). Regarding release numbers, 58.1% projects do not have any specific release recorded. 37.6% have less than 50 releases when only the rest 4.3% have more than 50 releases. All the previously mentioned data is always available from GitHub, and queries to the GitHub APIs will always return a valid information.

Regarding the adopted open source licenses, 23.7% projects did not specify the licenses they adopt. As for the other projects, MIT, Apache 2.0 and GNU GPL v3.0 are the most popular licenses with 53.2% projects adopting one of them (Fig. 6(h)). Therein, 13.6% projects adopted non-mainstream license (identified as ‘Other’).

We also validate the APIs of Reddit and StackOverflow by finding the amount of discussion threads on each of the 100k OSS projects. As shown in Fig. 5, 14.5% projects are generally discussed in Reddit with only 5.8% projects having more than 100 posts (Fig. 5(a)). On the other hand, 13.0% projects are discussed (raised technical questions) in StackOverflow (Fig. 5(b)). Therein, 3.6% projects have more than 100 questions raised on. In general, such results show that it is hard to find sufficient generic or technical discussion regarding specific OS projects from Reddit and StackOverflow.

Based on the interview results, especially the obtained factors that are considered important by the practitioners (shown in Table 2), we further validate whether such factors can be analyzed via the automatically obtained data from the previously mentioned portals with the results shown in Table 4.

According to the investigation on automatic OSS data extraction on the Top 100k OSS projects on Github via the APIs of



(a) OSS Project Stats by Reddit Post Number (b) OSS Project Stats by StackOverflow Questions

Fig. 5. Stats for Top 100k Github Projects in Reddit and StackOverflow.

the five public portals, we find that the majority of the metrics towards the *Community Support and Adoption* factor can be automatically done via data extractions from such APIs. To be noted, regarding *Communication*, the question and answer sources (i.e., StackOverflow) contain information on limited number of OSS projects, which limits the availability towards the evaluation on such category. In addition, we also find, despite the availability of automatic data extraction and measuring via APIs, many of the metrics require further calculation and learning, as well as multiple queries to obtain. For example, in order to measure the *Number of Independent Developers*, we must get the list of contributors of a particular project via multiple queries first (max items per page for Github API is 100), and check the "Independentness" of each contributor via further investigating his/her organization status. Thus, such process shall be, to some extent, time-consuming, when the limit rate of the API usage shall be also taken into account.

Another category can be automatically measured is *License*, as shown in the data, 76.6% of the projects contain specific License information. On the other hand, for the other categories, automatic data extraction and full-grained evaluation is hindered by the limited availability of the according data, as only very limited percentage of the metrics can be automatically done via data extraction (shown in Table 2). And amongst these categories, the evaluation of *Maturity* category depends on the availability of the release data from repository dataset, when for the obtained 100k projects only 42.2% provide such information. Measuring the availability of *Documentation* shall also depend on the information extracted from project description and homepage, while only around 40% of projects provide those.

As for the security vulnerability evaluation, the NVD Dataset provide information for 12838 projects (12.8%). However, it is not clear if the projects not reported in NVD do not contain security vulnerabilities at all, or simply are not indexed by the NVD dataset. However, it is important to note that the NVD performs analysis on CVEs published to the CVE Dictionary. Every CVE has a CVE ID which is used by cybersecurity product/service vendors and researchers for identifying vulnerabilities and for cross-linking with other repositories that also use the CVE ID. However, it is possible that some security vulnerabilities are not publicly reported with assigned CVE IDs.

As shown in Table 4, amongst the 170 metrics identified, 40 of them are potentially available to be extracted automatically. In addition, *License type* and *Development Language*, though seen as sub-factors, can be automatically measured as well. Therein, the number of automatic measurable metrics for all 100k projects (#full-auto) and that for part of them (#part-auto) are also shown. Only 22 metrics out of 170 can be obtained automatically for all 100k projects when the others are only available for part of the projects. In addition, 22 metrics require multi-queries to complete when 21 require further calculation and/or learning to determine.

5. Discussion

In this Section, we discuss the results of our RQs and we present the threats to validity of our work.

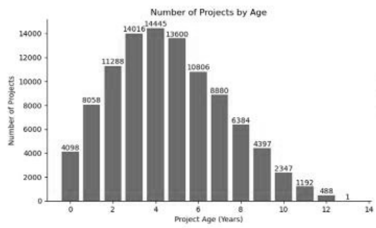
The factors and the metrics adopted to evaluate and select OSS (RQ1–RQ2) evolved over time. While in 2015 (Taibi, 2015; Lenarduzzi et al., 2020) factors such as *Customization easiness* and *Ethic* were the most important, nowadays we cannot state the same. Already in 2020 (Lenarduzzi et al., 2020) such factors have been incorporated inside other more valuable factors such as *Quality*, while today are not mentioned anymore. On the other side *License* and *Documentation*, which are the most mentioned factors nowadays were side factors in 2020. As a matter of fact the latter was a sub-section of *Development Process*, while both were not even considered in 2015. Moreover, ethical principles, that were very relevant in 2015, are not even mentioned in 2020.

Nowadays the trend is to search for OSSs which are ready to be integrated as is. In order to incorporate OSSs without falling into lawsuit particular attention needs to be put into the *License type*, while to guarantee the correct functioning the focus needs to be put in the documentation. A clear example is the necessity of a clear definition of the *system requirements* when incorporating libraries.

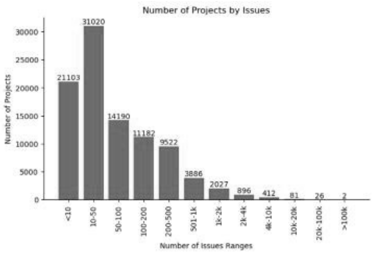
Another factor which gained a lot of importance over time is *Security*. While in 2015 was a factor with medium relevance, nowadays it is a keypoint for measuring quality of an OSS. The growing number of portals dedicated to ensure absence of vulnerabilities proves that people are concerned of the use of OSSs when embedded in their system. In particular they strongly rely on such portals to check the history of the OSSs to incorporate and in some cases also to ensure that proper reports are delivered when a new vulnerability is discovered.

Also, the importance of a factor is not necessarily proportional to the number of metrics. When specifying the importance of the assessment factors, we may see that some measurable factors are perhaps just eliminated, while some important factors cannot be automatically evaluated using the extracted information and require a manual assessment. Moreover, it is important to note that the lack of concrete metrics for some factors, such as community reputation or community sustainability might be because these factors are too abstract. Some researchers already addressed some of these aspects. As an example, Cai and Zhu (2016) and Hu et al. (2012) already proposed some metrics to evaluate the community reputation while Gamalielsson and Lundell (2014) also identified approaches for contributing to the community sustainability. However, these models are not yet diffused in industry, and this might be the reason why our interviewees were not aware of them.

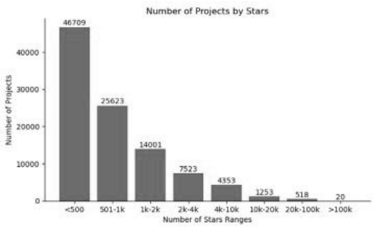
The source of information adopted to evaluate OSS (RQ3) did not change completely from the previous years. Users are



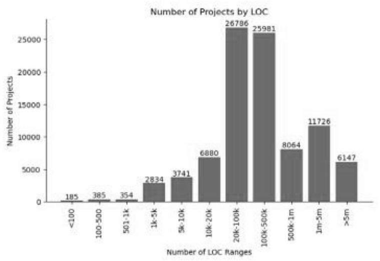
(a) OSS Project Stats by Age (Years)



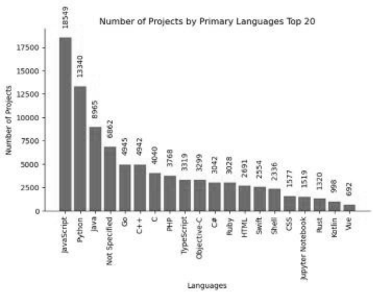
(b) OSS Project Stats by Issues



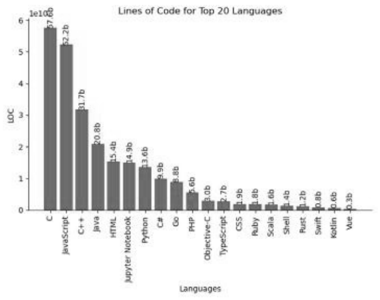
(c) OSS Project Stats by Stars



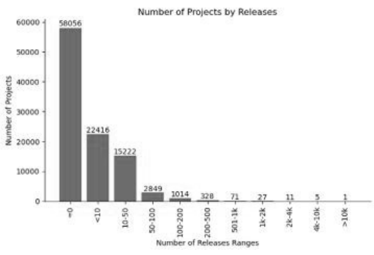
(d) OSS Project Stats by LOC



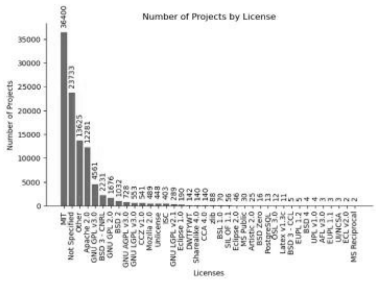
(e) OSS Project Stats by Top 20 Primary Languages



(f) Stats for Top 20 Language by LOC



(g) OSS Project Stats by Releases



(h) OSS Project Stats by Licenses

Fig. 6. General Stats for Top 100k Github Projects.

still adopting project repositories and issue trackers as main source of information. Moreover, an effect of the newly introduced factor security, is that now the selection also require security related information, that are commonly fetched from security databases such as the NIST NVD (Anon, 2020x), CVE (Anon, 2020y), and CWE (Anon, 2020z). In addition, many vendors like Synopsys (Anon, 2020j) and WhiteSource (Anon, 2020k) offer software composition analysis solutions that facilitate licence risk management, vulnerability identification and management, risk reporting, etc.

Unexpectedly, even nowadays, not all the portals can provide complete information for evaluating the information needed by the practitioners (RQ4). The analysis of the 100K most starred projects in GitHub showed that only the information coming from

Table 4
Factor information availability stats for top 100K projects.

Factor (#part-auto/#full-auto/#metrics)	Portal	#	%
Community Support and Adoption (5/16/41)			
Popularity	R		
Number of Watch	R	100000	100.00%
Number of Stars	R	100000	100.00%
Number of Forks	R	100000	100.00%
Number of Downloads	R	42260	42.26%
Community reputation	I+*		
Fast response to issues (△)	I+*	100000	100.00%
Community size	R+*		
Number of Contributors	R*	100000	100.00%
Number of Subscribers	R	100000	100.00%
Community age	R+	100000	100.00%
Number of Involved developers per company	R+*	100000	100.00%
Number of Independent developers	R+*	100000	100.00%
Community support	R+*, Q+*		
Activeness (△)	R+*	100000	100.00%
Responsiveness (△)	R+*, Q+*	14474	14.47%
Communication	I+*, Q+*		
Availability of questions/answers	I, Q	100000	100.00%
Availability of forum	Q	14474	14.47%
Responsiveness of postings	I+*, Q+*	14474	14.47%
Quality of postings (△)	Q+*	14474	14.47%
Sustainability	I+*		
Existence of maintainer	I+*	100000	100.00%
Product Team	R+*		
Developer quality (△)	R+*	100000	100.00%
Developer Productivity (△)	R+*	100000	100.00%
Responsiveness	I+*		
Avg. bug fixing time	I+*	100000	100.00%
Avg time to implement new issues	I+*	100000	100.00%
Documentation (3/0/12)			
Usage documentation	R*, F+		
Availability of updated documentation (△)	R*	42260	42.26%
Availability of documentation/books/online docs (△)	R	39499	39.50%
Availability of Tutorial or Examples (△)	R	39499	39.50%
License (1/0/9)	R		
License type	R	76576	76.58%
Operational SW Characteristics (1/0/9)	R		
Development language	R	93148	93.15%
Maturity (4/3/11)	R+*		
Number of forks	R	100000	100.00%
Release frequency	R+*	42260	42.26%
Number of releases	R	42260	42.26%
Age (in Years)	R+	100000	100.00%
Number of commits	R	100000	100.00%
Development versions	R*	42260	42.26%
New feature integration	R+*	42260	42.26%
Quality (5/3/47)			
Reliability	I+*		
Average bug age	I+*	100000	100.00%
Avg. bug fixing time	I+*	100000	100.00%
Security	R+*, Q+*, S+*		
Number security vulnerabilities	S	12838	12.84%
Number of Vulnerabilities reported on the NVD portal	S	12838	12.84%
Vulnerability Resolving time (△)	R+*, S+	12838	12.84%
Community concern towards security (△)	R+*, Q+*, S+*	12838	12.84%
Vulnerability impact (△)	S	12838	12.84%
Code Quality	R		
Code size	R	100000	100.00%
Trustworthiness (1/0/22)	R+*		
Consistent release updates pace (△)	R+*	42260	42.26%

Repository(R), Issue Tracker(I), Questions and Answer portals(Q), Forum & Blogs(F), Security(S)
Calculation Required(+), Multi-queries Required(*), Proxy Metrics(△).

the project repositories (e.g GitHub) are always available, except for the license information (76.6%). Considering other factors such as the communication, the situation does not improve, and only in 14.5% of projects we were able to automatically extract the relevant information from their APIs. For example, it is noticeable that the APIs of StackOverflow enable the extraction of other information, e.g., the textual content of questions and answers, the users' reputation, and so on. However, it requires further learning and calculation to elicit additional information from such textual content. The possibilities towards such directions shall be studied further in our future studies. Furthermore, some of the Github views, though providing valuable information but being inaccessible directly from APIs, (e.g., the GitHub insight view) are not covered herein. Due to the diversity of application domains, organizational needs, and constraints, practitioners in different organizations may explain the factors from their own perspective and may adopt different metrics in the OSS evaluation. A good example are the metrics associated with the factor of *Maturity*. Metrics such as the number of releases and the system growth in the roadmap were commonly concerned in the evaluation of software maturity; besides, some practitioners also took the number of commits and the number of forks as important metric in the evaluation. The total number of commits itself might not be enough when evaluating software maturity. The prevalence of commits over time and the types of commits could be additional and useful information in the evaluation. Therefore, how the metrics help with the evaluation of the factors could have been clarified.

The results of this work show a discrepancy between the information required by the practitioners to evaluate OSS and the information actually available on the portals, confirming that the collection of the factors required to evaluate an OSS project is very time-consuming, mainly because most of the information required is not commonly available on the OSS project (Kamei et al., 2018; Lenarduzzi et al., 2020; Del Bianco et al., 2010).

The automation of the data extraction, using portal APIs might help practitioners reducing the collection time and the subjectivity. The result of this work could be highly beneficial for OSS producers, since they could check if they are providing all the information commonly required by who is evaluating their products, and maximize the likelihood of being selected. The result can also be useful to potential OSS adopters, who will speed-up the collection of the information needed for the evaluation of the product.

Even in case OSS producers do not enhance their portals by providing the information required by the practitioners to assess OSS, the results of this work could be useful for practitioners that need to evaluate an OSS product. The list of factors can be effectively used as checklist to verify if all the potentially important characteristics of OSS have been duly evaluated. For instance, a practitioner could have forgotten to evaluate the trend of the community activity and he/she could adopt an OSS product that has a "dissolving" community: this could create problems in the future because of the lack of maintenance and updates. The usage of checklist would allow practitioners to double check if they considered all factors, thus reducing the potential unexpected issues that could come up after the adoption.

5.1. Future research directions

As a result of our findings, we propose the following directions for future research in this area.

Focus on the definition of a common tool to automatically extract information needed for the evaluation of OSS, investigating proxy metrics in case direct metrics are not available.

Definition of refined and customizable models (which may be obtained by merging multiple available approaches) and favor its adoption through rigorous and extensive validation in industrial settings. This could increase the validity of the model and thus its dissemination in industry, where OSS is still not widely adopted. Several models already exist but, according to the results of our previous literature review (Lenarduzzi et al., 2020), they have not been strongly validated and, as a consequence, adoption has been limited.

Try to target the models at quality factors that are of real interest for stakeholders. Most of the available models focus on the overall quality of the product, but few of them are able to adequately assess each single factor that composes the overall quality of the OSS product. This can complicate the assessment of OSS products by stakeholders, who are interested in specific quality factors: e.g., developers are likely more interested in reliability or testability aspects, while business people may be more interested in cost or maintenance factors, etc..

In the studies, we identified 170 metrics to measure the factors for OSS evaluation and selection, based on which we shall conduct an in-depth analysis to gain a better understanding of the rationale for the metrics. The rationale explains why a metric helps gain insights into the factors, and the assumption or other information useful in evaluating an OSS. It helps to identify the needed data to extract from the available portals and to automate the OSS analysis and assessment process. With the explanation of why the metrics are needed, practitioners can also better understand the factors and their assessment, which further eases the process to adopt the OSS selection models and tools in the software development practice.

Furthermore, besides the common evaluation metrics identified in this study that suits targeting any OSS, it is noticeable that the domain fitness of such targeting OSS is also of great importance. Though this study focuses on the general quality attributes of OSS, the assessment of domain fitness should be taken into account with the domain-fitting OSS candidates limited so that the evaluation effort can be largely reduced.

Develop tools that support the research directions listed above (i.e., tools able to support and simplify the applicability of the proposed models during the evaluation of OSS products).

Disseminate the information that should be provided on OSS portals, so as to enable OSS producers to consider them as part of their marketing and communication strategies (Del Bianco et al., 2012).

5.2. Threats to validity

We applied the structure suggested by Yin (2009) to report threats to the validity of this study and measures for mitigating them. We report internal validity, external validity, construct validity, and reliability.

Internal Validity. One limitation that is always a part of survey research is that surveys can only reveal the perceptions of the respondents which might not fully represent reality. However, our analysis was performed by means of semi-structured interviews, which gave the interviewers the possibility to request additional information regarding unclear or imprecise statements by the respondents. The responses were analyzed and quality-checked by a team of four researchers.

External Validity. Overall, a total of 23 practitioners were interviewed. We considered only experienced respondents and did not accept any interviewees with an academic background. However, we are aware that the convenience sampling approach we adopted could be biased, even if we tried to maximize the diversity. For example, practitioners from different domains, such as those developing real-time or safety-critical systems, might have provided a different set of answers. As for the projects we selected to validate the presence of information in OSS portals, we are aware that the 100K most starred GitHub projects might not represent the whole OSS ecosystem, but we believe they might be a good representative of them. We also think that less popular projects, might only perform worst than the selected ones.

We therefore think that threats to external validity are reasonable. However, additional responses and additional projects should be analyzed in the future.

Construct Validity. The interview guidelines were developed on the basis of the previously performed surveys (Del Bianco et al., 2009; Taibi, 2015). Therefore, the questions are aligned with standard terminology and cover the most relevant characteristics and metrics. In addition, the survey was conducted in interviews, which allowed both the interviewees and the interviewer to ask questions if something was unclear.

Reliability. The survey design, its execution, and the analysis followed a strict protocol, which allows replication of the survey. However, the open questions were analyzed qualitatively, which is always subjective to some extent, but the resulting codes were documented.

Table A.1
Results from the interviews.

Factor		Measure	#	Median
Community support and adoption			10	4.5
	Popularity		9	3
		# Watch	4	3
		# Stars	13	3
		# Fork	4	3
		# Downloads	13	3
	Community reputation		11	3
		Member of a foundation	1	4
		Complete administration mechanism	1	5
		Fast response to issues	1	5
	Community size		13	3
		# Contributors	11	4
		# Subscribers	3	3
		Community age	12	3
		# Involved developers per company	3	3
		# Independent developers	3	3
		Activeness	3	3
		Responsiveness	2	3.5
	Communication		6	3.5
		Availability of questions/answers	11	3
		Availability of forum	4	2.5
		# Mailing lists	3	3
		Traffic on the mailing list	3	3
		Responsiveness of postings	4	4
		Friendliness	6	2.5
		Quality of postings	3	3
	Involvement		9	3
		Clear project management	1	5
	Sustainability		11	3
		Existence of maintainer	11	3
	Product Team		5	3
		Developer quality	3	4
		Developer Productivity	2	3
	Responsiveness		1	5
	Scheduled updates		1	2.5
	Fast respond to user's needs		1	2.5
Documentation			14	4
	Avail. of documentation/books/online docs		5	3
		Avg time to implement new issues	1	4
		Avail. of updated documentation	9	4
	Avail. of development process documentation		4	3
		Avail. of getting started tutorial	1	5
	Avail. of Tutorial or Examples		5	5
	Usage documentation		4	4
	Avail. of best practices		4	4
			4	3
	Software requirements		11	3
		Complete doc. on SW requirements	1	5
	Hardware requirements		8	3.5
		Complete doc. on HW requirements	1	5
	Roadmap		7	3
	Test case documentation		4	3
License			21	4
		License type	20	5
		Law conformance	9	5
		License Compatibility	10	3
		OSS obligation fulfillment	1	5
		Existence of malicious OS obligation	1	5
		Contagiousness	1	5
		Multiple license option	1	2.5
		Dual License with limited features	7	4

(*continued on next page*)

This work was based on information extracted from OSS portals and the available APIs, and therefore, reliability of the assessment depends partly on the availability and reliability of the portals. Some projects might be managed and discussed on different platforms like the different issue tracking systems, the extracted information from the available APIs might be incomplete, which may affect the assessment results. On the other hand, we identified from the interviews the most used portals in each category of the sources of information. This helps mitigate the threat to some extent. Moreover, some projects might serve the purpose of providing resources, and not source code. However, Github API does not provide filtering functions towards excluding such projects. We believe that, this threat could be mitigated by the large amount of projects we selected.

Table A.1 (*continued*).

Factor		Measure	#	Median
Operational SW characteristics			6	4
	Trialability		5	3
		Available for independent verification and compile	1	5
		Provide demo for quick evaluation	1	4
	Independence from other SW		11	3
		Run independently	1	5
		Supports independent libraries	1	5
		Fewer dependences	7	4
	Development language		5	4
		Mainstream dev Lang	4	4
		Language know in the company	2	4.5
		Programming language uniformity	5	4
	Multiplatform support		5	3
	Standard compliance		5	4
		Testability	6	3.5
Maturity			6	3.5
		# forks	3	3
		Stability	7	5
		Release version stability	1	5
		# releases	10	4
		Release frequency	7	4
		# releases	3	4
		Age (#Years)	4	4
		# commits	3	3
		Development versions	6	4
		System growth	9	4
		New feature integration	1	5
Risk (Perceived risks)			7	4.5
	Perceived lack of integrity		5	3
	Perceived high availability		5	4
		Test according to context	1	4
		Analysis and pre-examination	1	5
		Comply with business requirements	1	5
	Strategic risks		5	3
		Influence of operation specified	1	4
	Hazard risks		5	4
		consequences specified	1	5
		Code security	1	4
		Virus scanning	1	4
		Risk of no maintenance	1	2.5
Quality			6	3.5
	Reliability		3	4
		Component reliability	2	5
		Architecture reliability	7	4
		System reliability	2	4.5
		# Bugs (open, closed, …)/bug density	8	4
		Average bug age	2	4.5
		Mean time between software failure (MTBF)	8	4
	Performances		4	4.5
		Main functionality external performance standards	1	5
		Based on business	1	2.5
		Construct verification environment	1	2.5
		Comparison with similar software	1	2.5
	Security		15	4
		# security vulnerabilities	12	3
		#Vulnerabilities reported on the NVD portal	14	4
		Security report	7	5
		Vulnerability Resolving time	2	5
		Community concern towards security	1	5
		Vulnerability impact	1	5
	Modularity		3	3
		Select OSS based on module	1	5
	Portability		3	4
		Adaptability	2	4.5
		Installability	2	4.5
	Flexibility/Exploitability		3	3
		Support usage patterns	1	2.5
		Reasonable function wrapper	1	2.5

(*continued on next page*)

Table A.1 (*continued*).

Factor		Measure	#	Median
	Code Quality		13	4
		Code complexity (class, methods, ..)	10	3
		Change proneness	3	3
		Fault proneness	3	4
		Test coverage	4	4.5
		Code size	7	3
		Technical difficulty	3	4
	Coding conventions		9	3
		Usage of linters for checking coding conventions compliance	7	3
	Maintainability		3	4
	Testability		2	4
	Changeability		2	3.5
	Update/Upgrade/Add-ons/Plugin		3	4
		Update capability between versions	1	2.5
		Easy to update to new version	5	3
		API compatibility between versions	1	2.5
	Architectural quality		5	3
Trustworthiness			6	4
	Component		4	3.5
		Functionality	1	5
	Architecture		4	3
		Difference with reality	1	4
	System		4	3.5
		Percentage of system failure	1	5
	OSS provider reputation		4	3.5
	Existence of benchmark/test		4	3.5
		Fast responsiveness to malicious affairs	1	5
		Transparency	1	4
		Test cases availability	1	4
	Collaboration with other product		4	2.5
		Even distribution among code submitters	1	2.5
		Consistent release updates pace	1	2.5
		In-time vulnerability publishing	1	2.5
		Measure-related information (i.e. measure possibility)	1	2.5
	Assessment results from 3rd parties		2	3.75

6. Conclusion

In this paper, we investigated the factors considered by companies when selecting OSS to be integrated in the software they develop, and we analyzed their availability in the OSS portals, in particular using OSS portal APIs.

We identified a set 8 factors and 74 sub-factors, together with 170 metrics that companies commonly use to evaluate and select OSS. Unexpectedly, only a small part of the factors can be evaluated automatically, and out of 170 metrics, only 40 are available from project portals APIs.

The automated extraction of the information from the 100K most starred GitHub projects showed that only 22 metrics out of 40 returned information for all the 100K projects. 2 metrics returned information for around 80% of the projects while another 7 for around 40%. The other 4 metrics returned information for below 15%.

It is important to note that the extraction consider some of the most famous OSS projects. Therefore, we can speculate that the vast majority of less common and less used projects might provide even less information.

The result of this work enable us to create a list of updated factors and metrics, together with the list of automatically collectable ones, that practitioners can use to select OSS.

Results can be used also by researchers to further validate the factors and metrics, or providing frameworks or tools to ease the selection of OSS. Moreover, OSS producers, and repositories might also benefit of this results to understand which information they should provide from their APIs, so as to ease the evaluation of OSS projects, and increase the adoption likelihood.

Future work include the validation of the factors and metrics in industrial settings, reducing the subjectivity of the decisions. Moreover, we are planning to develop a tool and portal to automatically collect the information and enable the comparison of OSS projects, so as to ease the OSS selection phase.

CRediT authorship contribution statement

Xiaozhou Li: Software, Formal analysis, Investigation, Writing – original draft, Visualization. **Sergio Moreschini:** Software, Formal analysis, Investigation, Writing – original draft, Visualization. **Zheying Zhang:** Conceptualization, Investigation, Writing – review & editing. **Davide Taibi:** Conceptualization, Methodology, Funding acquisition, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Results from the interviews

See Table A.1.

References

Allen, J., Collison, S., Luckey, R., 2009. Ohloh web site API.
Anon, 2020a. Flossmole. https://flossmole.org. (Accessed 30 December 2020).

Anon, 2020b. Free/libre open source software metrics. https://www.flossmetrics.org. (Accessed 30 December 2020).
Anon, 2020c. GH archive. https://www.gharchive.org. (Accessed 30 December 2020).
Anon, 2020d. GH torrent portal. https://ghtorrent.org. (Accessed 30 December 2020).
Anon, 2020e. Sourge forge research data. https://www3.nd.edu/~oss/Data/data.html. (Accessed 30 December 2020).
Anon, 2020f. OpenHub. https://www.openhub.net. (Accessed 30 December 2020).
Anon, 2020g. FlossHub. https://flosshub.org. (Accessed 30 December 2020).
Anon, 2020h. Candoia. http://candoia.github.io. (Accessed 30 December 2020).
Anon, 2020i. Repograms. https://github.com/RepoGrams/RepoGrams. (Accessed 30 December 2020).
Anon, 2020j. Synopsys - EDA tools, semiconductor IP and application security solutions. https://www.synopsys.com/. (Accessed 30 December 2020).
Anon, 2020k. Whitesource. https://www.whitesourcesoftware.com. (Accessed 30 December 2020).
Anon, 2020l. Sonarcloud. https://sonarcloud.io. (Accessed 30 December 2020).
Anon, 2020m. Sonartype. https://www.sonatype.com/. (Accessed 30 December 2020).
Anon, 2020n. Whitehat security|application security platform. https://www.whitehatsec.com/platform/software-composition-analysis/. (Accessed 30 December 2020).
Anon, 2020o. Kiuwan - end-to-end application security. https://www.kiuwan.com/. (Accessed 30 December 2020).
Anon, 2020p. Software heritage. https://www.softwareheritage.org. (Accessed 30 December 2020).
Anon, 2020q. Promise. http://promise.site.uottawa.ca/SERepository/. (Accessed 30 December 2020).
Anon, 2020r. Fossid. https://fossid.com. (Accessed 30 December 2020).
Anon, 2020s. Boa. http://boa.cs.iastate.edu/boa//. (Accessed 30 December 2020).
Anon, 2020t. Stack exchange. https://stackexchange.com. (Accessed 30 December 2020).
Anon, 2020u. Reddit. http://reddit.com. (Accessed 30 December 2020).
Anon, 2020v. Github. https://www.github.com. (Accessed 30 December 2020).
Anon, 2020w. Stack overflow. https://stackoverflow.com. (Accessed 30 December 2020).
Anon, 2020x. National vulnerability database (NVD). https://nvd.nist.gov/general. (Accessed 30 December 2020).
Anon, 2020y. Common vulnerabilities and exposures (CVE). https://ossindex.sonatype.org/doc/cve. (Accessed 30 December 2020).
Anon, 2020z. Common weakness enumeration (CWE). https://cwe.mitre.org. (Accessed 30 December 2020).
Anon, 2021. Replication package. https://github.com/clowee/Exploring-Factors-and-Measures-to-Select-Open-Source-Software. (Accessed 07 January 2021).
Avgeriou, P., Taibi, D., Ampatzoglou, A., Arcelli Fontana, F., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, N., Pigazzini, I., Saarimäki, N., Sas, D.D., de Toledo, S.S., Tsintzira, A.A., 2020. An overview and comparison of technical debt measurement tools. IEEE Softw..
Battaglia, M., 2008. Convenience Sampling. Encyclopedia of Survey Research Methods. Londres: SAGE Publications.
Brown, W.J., Malveau, R.C., McCormick, H.W., Mowbray, T.J., 1998. AntiPatterns: Refactoring software, architectures, and projects in crisis. John Wiley and Sons, New York.
Bruntink, M., 2014. An initial quality analysis of the ohloh software evolution data. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 65.
Cai, Y., Zhu, D., 2016. Reputation in an open source software community: Antecedents and impacts. Decis. Support Syst. 91, 103–112.
Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20 (6), 476–493.
Del Bianco, V., Lavazza, L., Lenarduzzi, V., Morasca, S., Taibi, D., Tosi, D., 2012. A study on OSS marketing and communication strategies. In: Open Source Systems: Long-Term Sustainability. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 338–343.
Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., 2009. Quality of open source software: The qualipso trustworthiness model. In: Open Source Ecosystems: Diverse Communities Interacting. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 199–212.
Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., Tosi, D., 2010. The qualiSPo approach to OSS product quality evaluation. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/libre/open Source Software Research and Development. In: FLOSS, vol. 2010, New York, NY, USA, pp. 23–28.
Di Cosmo, R., Zacchiroli, S., 2017. Software heritage: Why and how to preserve software source code. In: IPRES 2017 - 14th International Conference on Digital Preservation. Kyoto, Japan, pp. 1–10.
Dixon, M., 2016. Methods and systems for generating software quality index. Google Patents, https://patents.google.com/patent/WO2009089294A3.
Duijnhouwer, F., Widdows, C., 2003. Open source maturity model. Capgemini Expert Letter.
Dyer, R., Nguyen, H.A., Rajan, H., Nguyen, T.N., 2013. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In: 2013 35th International Conference on Software Engineering. ICSE, pp. 422–431.
Dyer, R., Nguyen, H.A., Rajan, H., Nguyen, T.N., 2015. Boa: Ultra-large-scale software repository and source-code mining. ACM Trans. Softw. Eng. Methodol. 25 (1).
Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston, MA, USA.
Gamalielsson, J., Lundell, B., 2014. Sustainability of open source software communities beyond a fork: how and why has the LibreOffice project evolved? J. Syst. Softw. 89, 128–145.
Golden, B., 2008. Open source maturity model. Open Source Bus. Resour. 4–9, Copyright - Copyright Talent First Network May 2008; Document feature - Tables; Last updated - 2020-11-18; SubjectsTermNotLitGenreText - United States–US.
Gousios, G., 2013a. The GHTorrent dataset and tool suite. In: Proceedings Of the 10th Working Conference on Mining Software Repositories. In: MSR, vol. 13, IEEE Press, Piscataway, NJ, USA, pp. 233–236.
Gousios, G., 2013b. The GHTorrent dataset and tool suite. In: 2013 10th Working Conference on Mining Software Repositories. MSR, pp. 233–236.
Hu, D., Zhao, J.L., Cheng, J., 2012. Reputation management in an open source developer social network: An empirical study on determinants of positive evaluations. Decis. Support Syst. 53 (3), 526–533.
Jansen, S., 2014. Measuring the health of open source software ecosystems: Beyond the scope of project health. Inf. Softw. Technol. 56 (11), 1508–1519, Special issue on Software Ecosystems.
Kamei, Y., Matsumoto, T., Yamashita, K., Ubayashi, N., Iwasaki, T., Shuichi, T., 2018. Studying the cost and effectiveness of OSS quality assessment models: An experience report of Fujitsu QNET. IEICE Trans. Inf. Syst. 2744–2753.
Kilamo, T., Lenarduzzi, V., Ahoniemi, T., Jaaksi, A., Rahikkala, J., Mikkonen, T., 2020. How the cathedral embraced the bazaar, and the bazaar became a cathedral. In: Open Source Systems. Springer International Publishing, Cham, pp. 141–147.
Lenarduzzi, V., Taibi, D., Tosi, D., Lavazza, L., Morasca, S., 2020. Open source software evaluation, selection, and adoption: a systematic literature review. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, pp. 437–444.
Madey, G., 2008. The sourceforge research data archive (SRDA). http://zerlot.cse.nd.edu/.
McCabe, T.J., 1976. A complexity measure. IEEE Trans. Softw. Eng. 2 (4), 308–320.
Petrinja, E., Nambakam, R., Sillitti, A., 2009. Introducing the OpenSource maturity model. In: Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/libre/open Source Software Research and Development. In: FLOSS, vol. 09, IEEE Computer Society, USA, pp. 37–41.
Robles, G., Koch, S., GonZález-barahona, J.M., Carlos, J., 2004. Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. In: Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems. RAMSS, IET, pp. 51–56.
Robles, G., Steinmacher, I., Adams, P., Treude, C., 2019. Twenty years of open source software: From skepticism to mainstream. IEEE Softw. 36 (6), 12–15.
Rocco, J.D., Ruscio, D.D., Sipio, C.D., Nguyen, P.T., Rubei, R., 2021. Development of recommendation systems for software engineering: the CROSSMINER experience. CoRR abs/2103.06987.
Roveda, R., Fontana, F.A., Pigazzini, I., Zanoni, M., 2018. Towards an architectural debt index. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, pp. 408–416.
Rozenberg, D., Beschastnikh, I., Kosmale, F., Poser, V., Becker, H., Palyart, M., Murphy, G.C., 2016. Comparing repositories visually with repograms. In: Proceedings Of the 13th International Conference on Mining Software Repositories. MSR, pp. 109–120.
Sbai, N., Lenarduzzi, V., Taibi, D., Sassi, S.B., Ghezala, H.H.B., 2018. Exploring information from OSS repositories and platforms to support OSS selection decisions. Inf. Softw. Technol. 104, 104–108.
Semeteys, R., 2008. Method for qualification and selection of open source software. Open Source Bus. Resour..
Soto, M., Ciolkowski, M., 2009. The qualOSS open source assessment model measuring the performance of open source communities. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement. pp. 498–501.
Spadini, D., Aniche, M., Bacchelli, A., 2018. Pydriller: Python framework for mining software repositories. In: Proceedings Of the 2018 26th ACM Joint

Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 908–911.

Taibi, D., 2015. An empirical investigation on the motivations for the adoption of open source software. ISBN: 9781612084381.

Taibi, D., Lavazza, L., Morasca, S., 2007a. OpenBQR: a framework for the assessment of OSS. In: Open Source Development, Adoption and Innovation. Springer US, Boston, MA, pp. 173–186.

Taibi, D., Lavazza, L., Morasca, S., 2007b. OpenBQR: a framework for the assessment of OSS. In: Feller, J., Fitzgerald, B., Scacchi, W., Sillitti, A. (Eds.), Open Source Development, Adoption And Innovation. Springer US, Boston, MA, pp. 173–186.

Wasserman, A.I., Guo, X., McMillian, B., Qian, K., Wei, M.-Y., Xu, Q., 2017. Osspal: Finding and evaluating open source software. In: Balaguer, F., Di Cosmo, R., Garrido, A., Kon, F., Robles, G., Zacchiroli, S. (Eds.), Open Source Systems: Towards Robust Practices. Springer International Publishing, Cham, pp. 193–203.

Wasserman, A.I., Pal, M., Chan, C., 2006. The Business Readiness Rating: A Framework for Evaluating Open Source. Technical Report.

Wuetherick, B., 2010. Basics of qualitative research: Techniques and procedures for developing grounded theory. Can. J. Univ. Continuing Educ. 36.

Yin, R., 2009. Case Study Research: Design And Methods, (Applied Social Research Methods, Vol. 5), fourth ed.h SAGE Publications, Inc.

PUBLICATION II

OSSARA: Abandonment Risk Assessment for Embedded Open Source Components

Xiaozhou Li, Sergio Moreschini, Fabiano Pecorelli, and Davide Taibi

IEEE Software 39.4 (2022), pp. 48–53
DOI: https://doi.org/10.1109/MS.2022.3163011

OSSARA: Abandonment Risk Assessment for Embedded Open Source Components

Xiaozhou Li, Sergio Moreschini, Fabiano Pecorelli, and **Davide Taibi**, Tampere University

// Systems with unmaintained embedded open source software (OSS) components are vulnerable to severe risks. This article introduces the OSS Abandonment Risk Assessment model to help companies avoid potentially dire consequences. //

SOFTWARE NEEDS TO be continuously updated and maintained to continue being useful.[1] This is particularly true for open source software (OSS) components and libraries, which are more and more often integrated into large and complex systems. For companies developing long-term projects, all embedded OSS components should guarantee lengthy life expectancies and be maintained as long as systems are in service. Embedding abandoned OSS in critical systems could expose companies to severe risks. For example, new security vulnerabilities could be exploited, bugs and issues might never be resolved, and functions could become obsolete and inadequate for new environments. Metaphorically, systems embedding abandoned OSSs are like vehicles with rusted gears or human bodies with malignant tumors. Indeed, the abandonment of OSS components might produce a "domino effect" that results in the inoperability of full systems. The importance of such a statement is in the fact that even if a single embedded software component is unavailable, a whole project can be compromised.

In this respect, we were recently asked by a local branch of a global company, which operates in different domains and with more than 200,000 employees in 150-plus countries, to devise a methodology aimed at identifying components embedded in its software products that were the most likely to be abandoned soon. To meet the requirements, we designed the OSS Abandonment Risk Assessment (OSSARA) model, which we present in this article. The model aims to assess the abandonment risk of a software system through prediction for every embedded OSS component and the criticality that each component represents. With OSSARA, practitioners can monitor a system's risk level and choose to maintain or replace OSS components.

Related Work

During the past decade, researchers have been paying great attention

Digital Object Identifier 10.1109/MS.2022.3163011
Date of current version: 20 June 2022

to software sustainability. Samoladas et al.[2] successfully exploited survival analysis methods to predict the survivability of software projects. Businge et al.[3] analyzed the survivability of 1,447 versions of 467 Eclipse third-party plug-ins and classified them into two categories: those relying on stable dependencies and those with at least one potentially unstable dependency. They observed that plug-ins that use only stable dependencies are likelier to maintain a higher source compatibility rate through time. Coelho et al.[4] leveraged machine learning to build a model to identify unmaintained GitHub projects, based on a set of 13 process metrics, achieving promising results. Afterward, they presented an extended version of the work,[5] defining a metric to indicate how risky it would be to depend on a given GitHub project.

Valiev et al.[6] assessed open source Python projects' sustainability based on ecosystem-level factors, i.e., those describing interdependencies among packages. They calculated sustainability by the mean of dormancy, i.e., the period of inactivity for a project repository. The results indicated that the number of connections as well as the dependency network position are significant factors affecting the projects' sustainability. Later, Mujahid et al.[7] proposed a scalable approach that relies on the package centrality in an ecosystem to identify packages in decline. The results of an evaluation conducted on the Node Package Manager ecosystem showed strong prediction capabilities, thus indicating centrality as an important factor for forecasting project abandonment.

In previous work,[8] we investigated approaches to automate the evaluation of information from OSS projects, although we did not propose an assessment and risk model. In contrast to the related literature, we are introducing a method to calculate abandonment risk based on the probability of embedded system components losing maintenance support. Moreover, thanks to the assistance we received from our case company, our method is suitable for real industrial applications.

Software Composition Using OSS

Software composition via the adoption of components off the shelf (COTS) has long been considered an effective practice.[9] Despite the disadvantages of COTS in terms of uneven performance, a lack of evolution control, and insufficient interoperating capabilities, using the components enables practitioners to avoid "reinventing the wheel." OSS can be viewed as COTS since most of the embedded components, e.g., libraries and plug-ins, are usually integrated as is. The main advantages of OSS components are the open licenses, which usually permit access to the source code and, eventually, making extensions. Moreover, OSS is often accessible without paying license fees, thus reducing adoption costs.

When developing a software project, the most common practice is to integrate several components and combine them by writing custom code. The amount of custom code is usually minimal compared to the size of the components. Developing all components as custom software might require significant effort, not only for the process itself but also for maintenance. However, creating a system consisting of several OSS components introduces risks since the maintenance of each one is usually delegated to the developer community. There may be cases where the community does not continue the upkeep. Companies with integrated unmaintained OSS components need to find alternatives, either deciding to maintain the components themselves or replacing them.

OSSARA

We propose OSSARA to assess a system's abandonment risk on the basis of embedded OSS components. The abandonment risk is calculated based on 1) the likelihood of each component losing maintenance support during a certain period and 2) the importance that each component has for the main system, following the classic risk assessment notion $Risk = Prob(Loss) \times Size(Loss)$.[10] Figure 1 depicts the OSSARA process. Starting from a software system that embeds several OSS components (14 in the example), we first calculate for each component the abandonment probability in the given time and the weight (abandonment probability and weight are represented by colors and box sizes, respectively). Then, we combine these two pieces of information to calculate the risk that the main system will be abandoned within the considered period.

More formally, the overall abandonment risk R_a for a system $(R_a \in [0,1])$ that integrates k OSS components is calculated as follows:

$$R_a = \sum_{m=1}^{k} w(O_m) * r(O_m),$$

where $w(O_m) \in [0,1]$ represents the weight of the OSS component O_m and $r(O_m) \in [0,1]$ conveys the risk that O_m will be abandoned. The weight of a component $w(O_m)$ can be quantified by counting the number of invocations (e.g., the number of imports in the code).

Predicting OSS Abandonment

Identifying inactive and abandoned OSS could be easily performed by

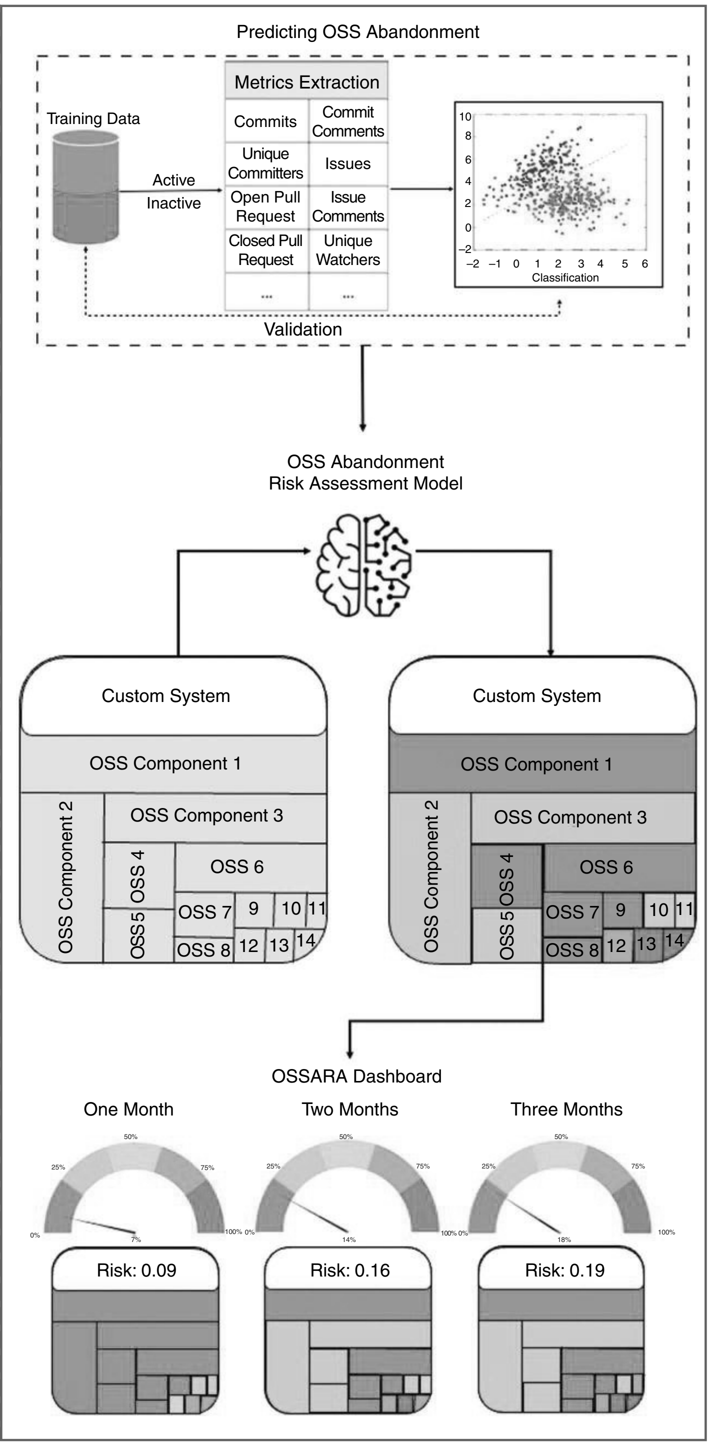


FIGURE 1. The OSSARA process.

directly checking for the presence of specific tags on SourceForge (https://sourceforge.net). However, by then, it would be too late for a company to find proper alternatives. Thus, it is necessary to find a way to foresee potential abandonment.

Predicting the abandonment risk for a software component is a multiconcern assurance problem since it could depend on several aspects, such as poor performance, insufficient maintainability, and so on. Commonly, an OSS component is considered abandoned based only on the number of commits performed on the system repository in a given time interval.[2,11,12] Therefore, one could trivially think to use this information as a sole predictor to foresee abandonment; i.e., if the number of commits on a certain project repository falls below a predefined threshold within a certain period, the component is considered abandoned. However, the process for determining the threshold or period length will vary among practitioners.

Our case company considers an OSS project abandoned if the software has not had any releases or commits within the previous six months, which is a comparatively stricter threshold than that suggested by Khondhu et al.[11] Furthermore, it is also possible that when an OSS community does not focus on committing, the contributors remain active in handling pull requests and discussing relevant issues. In this case, the suggestion from our case company is that measures regarding committing (e.g., the daily number of commits), communication (e.g., the daily issue comments), and issue handling activities (e.g., the daily closed pull requests) be taken into account. For these reasons, we propose to apply supervised techniques to predict the abandonment likelihood for an OSS component on the basis of key activities (e.g., commits, issues, pull requests, and so on). This would overcome the problem of subjective thresholds: rather than relying on predefined thresholds, prediction can be adapted to a component.

In detail, to predict the abandonment of an OSS component, we propose the following pipeline:

- *Step 1—Data crawling*: We gather data from all 125,486,232 GitHub projects by using the GHTorrent data set (https://ghtorrent.org). The selected metrics consist of the number of commits, commit comments, unique committers, issues, issue comments, watchers, and open and closed pull requests.
- *Step 2—Data preprocessing*: We create training data for each OSS project that fulfill the criteria specified by our case company, labeling projects active on the basis of 1) having more than 2,000 commits, 2) having more than 1,000 days of activity (from the day of creation to the final commit day), 3) having at least one commit in the past six months, and 4) having days with zero commits equal less than 50% of the days of activity. Furthermore, labeled training data are prepared based on the target prediction period (e.g., one, two, or three months), with their dimensions set to provide the best accuracy.
- *Step 3—Prediction*: Using the labeled and preprocessed data, we train the classifiers with the best performance for the target prediction periods. With the target OSS component data as input, the classifier predicts whether a component is active or abandoned. Its accuracy is used as the probability of the OSS being active or abandoned, as it indicates the likelihood that the prediction is correct.

Validation

To validate the proposed methodology, we conduct a preliminary evaluation on 12,208 OSS projects that contain at least 1,000 commits from at least five unique contributors and are watched by at least 100 users (the data set is shared at https://doi.org/10.6084/m9.figshare.16944001.v1). Such criteria ensure the popularity and longevity of the candidates. The data set is extracted from GHTorrent (until the 1 June 2019 dump) and labeled according to the preceding guidelines (see step 2). Among the four classification algorithms we selected, i.e., decision tree, support vector machine, logistic regression, and naive Bayes, we find that logistic regression has the highest accuracy with the data set. We also apply a 10-fold cross-validation strategy to assess the prediction capabilities of the model. The results of the validation indicate an F1 score of ≈ 0.86 (a ±0.01 estimated error), with the Matthews correlation coefficient being 0.73; hence, we conclude that the proposed methodology is reliable enough.

Working Example

This section presents a working example of the proposed method. Due to a nondisclosure agreement with our case company, we cannot provide details about the real industrial application of our technique. However, to demonstrate OSSARA at work, we apply it to an open source software project. We take Keras as an example of software developed in-house that needs to integrate various OSS components. Keras is an OSS project providing a Python-written deep learning application programming interface for TensorFlow libraries. For our analysis, we chose release 2.7.0 RC1 (https://github.com/keras-team/keras/releases/tag/v2.7.0-rc1), accessed on 26 October 2021.

We conduct our analysis by focusing only on the 536 Python files from the repository and detect the OSS components (i.e., packages) imported within each file. Furthermore, to ease the computation as well as the explanation of the results, we consider the 20 packages most frequently imported in Keras. We examine only the five most commonly imported OSS components in Keras, namely, TensorFlow, CPython, Numpy, abseil-py, and h5py. The packages, e.g., re, random, collections, and so on, belong to the CPython component and will be considered together. The weight of each component is calculated by the percentage of files importing it.

We conduct our analysis using three time frames to predict the abandonment risk of embedded OSS components in one, two, and three months. The same approach can be applied to longer periods. To simplify the process, we quantify the weight of each imported Python package by counting the number of imports across all the project files. Figure 2 summarizes the results. They show that for one month, all components are safe from abandonment. When considering a two-month time frame, the abseil-py package appears to have a high risk of being abandoned (i.e., 85.8%). Finally, for the three-month analysis, the risk that the h5py package will be abandoned rises. The three key components, namely, TensorFlow, Cpython, and Numpy, remain active throughout. Based on our calculation, the overall abandonment risk for Keras grows from 0.138 in one month to 0.215 in two months and 0.248 in three months. From the repository history of abseil-py and h5py, we observe that since 2020, both projects have had very low committing and issue handling rates from a small group of contributors, which legitimizes the risks assessment.

This example shows that OSSARA predicts the abandonment risk to software systems that embed OSS components. However, this oversimplified demonstration aims only to explain how our method works when probability-based conclusions might not reflect reality. Please note that we did not consider hierarchical relations. For example, the 20 selected packages might use other components that have a high abandonment risk. Meanwhile, although the model meets real industrial needs—its main strength—its generalizability can be limited. Furthermore, the application of risk prediction toward component replacement and integration requires the support of software engineering assessment and decision making.[13] Finally, other metrics might have different prediction power for abandonment risk.

Integrating abandoned OSS in software-intensive systems is hazardous and could result in severe consequences, which concerns practitioners. Especially when functions from abandoned components are integrated into highly critical modules, the consequences from abandoned OSS that lacks maintenance can be unbearable. To foresee such risks, we proposed OSSARA to provide an assessment and prediction pipeline. The model also supports continuous adaptation and customization through which practitioners can conduct optimized prediction via up-to-date OSS activity data, with selectively effective and even customized algorithms. The model was positively received by our case company, which is adopting and integrating it into its continuous integration/continuous deployment pipeline. Future work will explore other analysis techniques, the inclusion of different metrics in the prediction model, and the integration of various types of OSS risk assessment, e.g., security assessment, license compliance assessment, and so on, as well as dependency and hierarchy analysis.

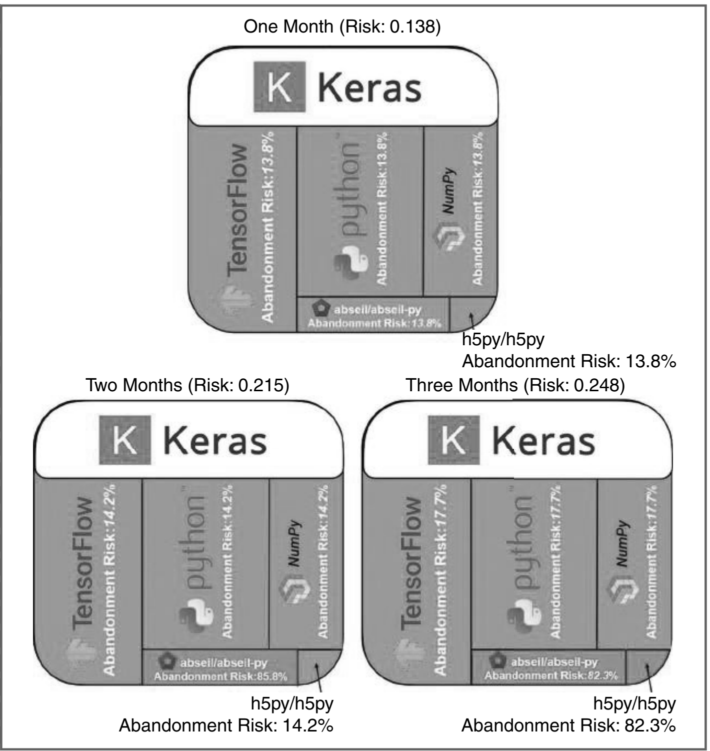


FIGURE 2. The abandonment risk assessment for Keras.

References

1. M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980, doi: 10.1109/PROC.1980.11805.
2. I. Samoladas, L. Angelis, and I. Stamelos, "Survival analysis on the duration of open source projects," *Inf. Softw. Technol.*, vol. 52, no. 9, pp. 902–922, 2010, doi: 10.1016/j.infsof.2010.05.001.
3. J. Businge, A. Serebrenik, and M. van den Brand, "Survival of eclipse third-party plug-ins," in *Proc. 2012 28th IEEE Int. Conf. Softw. Maintenance*

(ICSM), pp. 368–377, doi: 10.1109/ICSM.2012.6405295.
4. J. Coelho, M. T. Valente, L. L. Silva, and E. Shihab, "Identifying unmaintained projects in GitHub," in *Proc. 12th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2018, pp. 1–10, doi: 10.1145/3239235.3240501.
5. J. Coelho, M. T. Valente, L. Milen, and L. L. Silva, "Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects," *Inf. Softw. Technol.*, vol. 122, p. 106,274, Jun. 2020, doi: 10.1016/j.infsof.2020.106274.
6. M. Valiev, B. Vasilescu, and J. Herbsleb, "Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PYPI ecosystem," in *Proc. 2018 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, pp. 644–655, doi: 10.1145/3236024.3236062.
7. S. Mujahid, D. E. Costa, R. Abdalkareem, E. Shihab, M. A. Saied, and B. Adams, "Towards using package centrality trend to identify packages in decline," 2021, *arXiv:2107.10168*.
8. X. Li, S. Moreschini, Z. Zhang, and D. Taibi, "Exploring factors and metrics to select open source software components for integration: An empirical study," *J. Syst. Softw.*, vol. 188, p. 111,255, Jun. 2022, doi: 10.1016/j.jss.2022.111255.
9. B. Boehm and C. Abts, "COTS integration: Plug and pray?" *Computer*, vol. 32, no. 1, pp. 135–138, 1999, doi: 10.1109/2.738311.
10. B. Boehm, "Software risk management," in *Proc. Eur. Softw. Eng. Conf.*, Springer-Verlag, 1989, pp. 1–19, doi: 10.1007/3-540-51635-2_29.
11. J. Khondhu, A. Capiluppi, and K.-J. Stol, "Is it all lost? A study of inactive open source projects," in *Proc. IFIP Int. Conf. Open Source Syst.*, Springer-Verlag, 2013, pp. 61–79, doi: 10.1007/978-3-642-38928-3_5.
12. J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proc. 2017 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 186–196, doi: 10.1145/3106237.3106246.
13. R. A. Ribeiro, A. M. Moreira, P. Van den Broek, and A. Pimentel, "Hybrid assessment method for software engineering decisions," *Decis. Support Syst.*, vol. 51, no. 1, pp. 208–219, 2011, doi: 10.1016/j.dss.2010.12.009.

ABOUT THE AUTHORS

XIAOZHOU LI is a postdoctoral researcher in the Faculty of Information Technology and Communication Sciences, Tampere University, Tampere, 33720, Finland, where he is a researcher in the Cloud Software Evolution and Assessment group. Li received his Ph.D. in computer science from Tampere University. His research interests include open source software quality, software maintenance and evolution, and user review opinion mining. Contact him at xiaozhou.li@tuni.fi.

SERGIO MORESCHINI is a Ph.D. candidate in the Faculty of Information Technology and Communication Sciences, Tampere University, Tampere, 33720, Finland, where he is a researcher in the Cloud Software Evolution and Assessment group. His research interests include extended light field reconstruction for continuous parallax content. Contact him at sergio.moreschini@tuni.fi.

FABIANO PECORELLI is a postdoctoral researcher in the Cloud Software Evolution and Assessment group, Tampere University, Tampere, 33720, Finland. His research interests include software code and test quality, predictive analytics, and mining software repositories. Pecorelli received his M.S. in computer science from the University of Salerno, Italy. Contact him at fabiano.pecorelli@tuni.fi.

DAVIDE TAIBI is an associate professor at Tampere University, Tampere, 33720, Finland, where he heads the Cloud Software Evolution and Assessment research group. His research interests include empirical software engineering applied to cloud-native systems. Taibi received his Ph.D. in computer science from University of Insubria. He is a member of the International Software Engineering Network. Contact him at davide.taibi@tuni.fi.

PUBLICATION III

Cloud Continuum: The Definition

Sergio Moreschini, Fabiano Pecorelli, Xiaozhou Li, Sonia Naz, David Hästbacka, and Davide Taibi

IEEE Access 10 (2022), pp. 131876–131886
DOI: https://doi.org/10.1109/ACCESS.2022.3229185

Received 26 October 2022, accepted 10 December 2022, date of publication 14 December 2022, date of current version 22 December 2022.
Digital Object Identifier 10.1109/ACCESS.2022.3229185

SURVEY

Cloud Continuum: The Definition

SERGIO MORESCHINI[1], (Student Member, IEEE), FABIANO PECORELLI[1], (Associate Member, IEEE), XIAOZHOU LI[1], (Associate Member, IEEE), SONIA NAZ[1], DAVID HÄSTBACKA[1], (Member, IEEE), AND DAVIDE TAIBI[1,2], (Member, IEEE)
[1]Computing Sciences, Faculty of Information Technology and Communication Sciences, Tampere University, 33720 Tampere, Finland
[2]Empirical Software Engineering in Software, Systems and Services (M3S), University of Oulu, 90570 Oulu, Finland
Corresponding author: Sergio Moreschini (sergio.moreschini@tuni.fi)

This work was supported in part by the Ulla Tuominen Foundation (Finland), and in part by the Academy of Finland under Grant 349488-MuFAno.

ABSTRACT The cloud continuum concept has drawn increasing attention from practitioners, academics, and funding agencies and been adopted progressively. However, the concept remains mired in various definitions with different studies providing contrasting descriptions. Therefore, to understand the concept of cloud continuum and to provide its definition, in this work we conduct a systematic mapping study of the literature investigating the different definitions, how they evolved, and where does the cloud continue. The main outcome of this work is a complete definition that merges all the common aspects of cloud continuum, which enables practitioners and researchers to better understand what cloud continuum is.

INDEX TERMS Cloud continuum, edge, Fog.

I. INTRODUCTION

The adoption of service-oriented architecture in cloud computing has profoundly changed the way how software, especially large-scale distributed systems, are built [24]. The cloud is often viewed as an endless pool of resources, on which we build and scale applications for various purposes. Modern cloud systems, however, are inherently complex spanning public cloud to private cloud, possibly co-located across different regions, and may also include components and compute resources at the edge of the network.

Cloud continuum is one of the most recent hypes in the cloud computing domain and has raised interests of funding agencies of EU and US [1], [2], [3]. However, while the hype is increasing, its definition is still not clear, and various papers are describing the concept of cloud continuum inconsistently.

In order to understand the differences between the disparate definitions of cloud continuum, we propose a systematic mapping study of the literature.

In this work, we investigate the existing definitions and common characteristics of ''cloud continuum'' as well as their evolution through the time.

We formulate three main Research Questions (RQs) as follows.

- ***RQ1:*** *What are the definitions of cloud continuum?*
  With this RQ we aim at understanding whether there are different definitions of cloud continuum.
- ***RQ2:*** *How has the definition of cloud continuum evolved?*
  Via the comparison amongst the different definitions, we shall observe the changes from the earliest to the latest. In this way, we shall identify what are the new aspects taken into account regarding ''cloud continuum''.
- ***RQ3:*** *Where does the cloud continue?*
  As cloud is ''continued'' into other infrastructures, we expect to find cloud-to-* extensions, where * could be on premise servers, but also edge, or other infrastructures. In this RQ we aim at understanding which are these extensions, so as to clarify where the cloud could be continued.

The remainder of this paper is structured as follows. Section 2 presents related reviews. Section 3 describes the research method adopted. Section 4 presents the results answering the RQs. Section 5 discusses the results while Section 6 draws the conclusion and highlights future works.

II. BACKGROUND AND RELATED WORK

A. CLOUD, FOG, EDGE, AND MORE

Cloud computing builds on the promise of economies of scale in leveraging scalability and reliability. Scaling up is

The associate editor coordinating the review of this manuscript and approving it for publication was Rodrigo S. Couto.

made possible by creating multiple compute instances and distributing them. Containers have long been the basis for implementing microservices based architectures but recent advancement towards serverless and Functions as a Service further emphasize the role of the cloud as a platform abstracting underlying infrastructure resources [6], [18].

Fog computing can be simplified as the cloud brought closer to the use case applications. Fog nodes minimize load on the cloud and are able to host some services from the cloud, and thus respond faster and also reduce networking to the cloud [9]. Chiange et al. [10] define that ''fog is inclusive of cloud, core, metro, edge, clients, and things,'' and ''fog seeks to realize a seamless continuum of computing services from the cloud to the things'' instead of independent application resource pools.

Edge computing takes place at the edge of the network close to IoT devices, however, not necessarily on the IoT devices themselves but as close as one hop to them [25]. Edge computing has been pushed heavily by the telecommunication industry but it has also emerged from the need to perform computation closer the applications or with independence from cloud computing. Edge computing is characterised by short latency in contrast to cloud computing where transmission of data, allocation of resources typically includes delays.

For applications where large amounts of data needs to be processed both fog and edge computing can introduce benefits as cost savings in transfer, storage and processing. This includes, for example, data from thousands of sensors, audio and video streams, and emerging machine learning (ML) based solutions. In Virtual Reality (VR) and Augmented Reality (AR) edge computing together with low latency communication is claimed to enable cutting the cord, and it has been shown to achieve minimum gains of up to 30% reduction in end-to-end delay and even more for most parts of the communication [11].

B. RELATED WORK

Over the last few years, more and more researchers have been focusing on the cloud continuum paradigm. Therefore, some surveys/reviews on the subject have already been presented. In the following, we report an overview of the most relevant works available in the literature and discuss the differences with our work.

Al-Sharafi et al. [4] presented a literature review on the adoption of cloud computing services at the organizational level, with a focus on the elements that contribute to long-term adoption.

Pahl et al. [19] performed a literature review to identify, catalog, and compare the corpus of existing research on containers, their orchestration, and particularly the use of this technology in the cloud.

Bittencourt et al. [8] presented a literature review on IoT-Fog-Cloud continuum with the aim of understanding (i) what are the best types of infrastructures to deploy the entire ecosystem, (ii) what are the required mechanisms to allow orchestration, data exchange, and resource management, and (iii) what are the types of applications that can benefit most from this ecosystem.

Nguyen et al. [17] surveyed the current landscape of the existing approaches and tools that attempt to cope with this edge and cloud heterogeneity, scalability and dynamicity.

Bendechache et al. [7] surveyed the list of suitable methods, algorithms, and simulation approaches for resource management in cloud-to-thing continuum.

Ramanathan et al. [21] conducted a survey to retrieve all the resource allocation techniques that have been developed for the cloud continuum.

Svorobej et al. [23] reviewed the orchestration mechanisms along the cloud-to-thing continuum with a focus on container-based orchestration and orchestration architectures tailored for fog.

Asim et al. [5] provided a summary of research issues in Cloud computing and Edge computing, as well as current developments in resolving them with CI approaches.

Ghobaei-Arani et al. [13] provided a literature analysis aiming to identify the state-of-the-art mechanisms on resource management approaches in the fog computing environments.

Kampars et al. [14] reviewed application layer protocols that can be used for the communication between the IoT, edge and cloud layers.

Spataru [22] surveyed the applications of Blockchain or Smart Contracts for computing resources management, data storage, and services operation in the context of Cloud continuum.

Kansal et al. [15] presented a systematic literature review of the resource management approaches in fog/edge paradigm.

Compared to our work, the previous literature reviews spent a noticeable effort in understanding technical and managerial aspects of the cloud continuum (Table 1). Instead, our work focuses on identifying the definition of the cloud continuum, how it evolved, and where the cloud continues.

III. RESEARCH METHODOLOGY

In this study, we conducted a systematic mapping study of the literature, by taking into account the guidelines proposed by Petersen et al. [20]. The main aim was to systematically and impartially summarize and classify the collected information regarding the research questions. Specifically herein, we aimed to not only characterize all the existing definitions of the ''cloud continuum'' and other relevant concepts, but also to investigate the evolution of such definitions through time.

The process of the study included four main steps. Firstly, we established the research questions. Secondly, we defined the search strategy. Thirdly, we defined the data extraction strategy. Fourthly, we synthesized and visualized the obtained results.

TABLE 1. Summary of the related literature reviews.

Source	Target	Year
Al-Sharafi et al. [4]	Adoption of cloud computing services at the organizational level	2017
Pahl et al. [19]	Containers orchestration and usage in the Cloud	2017
Bittencourt et al. [8]	IoT-Fog-Cloud continuum infrastructures, orchestration, data exchange, and resource management	2018
Nguyen et al. [17]	Existing approaches and tools supporting edge and cloud development	2019
Bendechache et al. [7]	Resource management in cloud-to-thing continuum	2020
Ramanathan et al. [21]	Resource allocation techniques for the Cloud continuum	2020
Svorobej et al. [23]	Orchestration mechanisms along the cloud-to-thing continuum	2020
Asim et al. [5]	Research issues in Cloud computing and Edge computing	2020
ghobaei2020resource	Resource management approaches in the fog computing environments	2020
Kampars et al. [14]	Communication protocols between the IoT, edge and cloud layers	2021
Spartaru [22]	Blockchain usage in Cloud continuum	2021
Kansal et al. [15]	Resource management in fog/edge paradigm	2022
Our study	Definition of Cloud continuum	2022

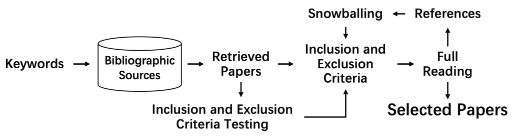


FIGURE 1. The Search and Selection Process.

TABLE 2. Initial search results by sources.

Library	Scopus	IEEE	ACM	WoS	Non-Duplicates
Count	271	148	61	102	378

A. SEARCH STRATEGY

The aim as well as the challenge for a systematic mapping study was to define the search query that enables the retrieval of a complete set of studies that contain the definitions [16]. For such a purpose, the search strategy encompassed a set of steps, namely, defining search string, identifying key sources, selecting primary studies, extracting data and synthesizing the results.

The search strategy involved the outline of the most relevant bibliographic sources and search terms, the definition of the inclusion and exclusion criteria, and the selection process relevant for the inclusion decision. Our search strategy can be depicted in Fig. 1.

As for the search terms, we included cloud concepts, Fog, Edge, and Continuum:

*(**cloud** AND (**edge** OR **fog**) AND **continuum**)*

We searched for scientific literature in four bibliographic sources: Scopus,[1] IEEEXplore Digital Library,[2] the ACM Digital Library,[3] and Web of Science.[4] The adoption of four databases ensured the completeness of the search results.

We conducted our search on March 1st 2022, retrieving 378 non-duplicated papers from the four sources. The number of papers retrieved for each source is reported in Table 2.

B. PRIMARY STUDIES SELECTION

In order to select the primary studies from the preliminary search results, we defined the inclusion and exclusion criteria shown in Table 3. We included the research papers published in journals or conferences, defining Cloud Continuum. On the other hand, we excluded the research papers that are not in English, duplicated, not discussing the topic connected to the defined research questions. Furthermore, we also excluded the papers that are not peer-reviewed, as well as the work plans or roadmap, posters and vision papers.

With the inclusion and exclusion criteria defined, we selected the primary studies via two steps. Firstly, two of the authors read the title and abstract of each paper separately to determine whether it should be excluded or be read fully. Whenever there was disagreement between them, a third person assert the decision by the inclusion and exclusion. Out of 378 papers screened, we had 93 disagreement with a Cohen's kappa coefficient of 0.51, indicating a moderate agreement [12]. As a result, we identified 181 papers that need to be considered for the next step.

We then ran a snowballing process including all the papers referenced by the 181 papers. We then followed the same process by applying inclusion and exclusion criteria to their titles and abstracts. As a result, we included two more papers: one peer-reviewed, and one grey literature [SP1]. The reason for including this specific non-peer-reviewed work [SP1] is due to its large amount of citations; especially when many of our selected papers referred to it as the first definition of cloud continuum. Though belonging to the gray literature, this study represents an important milestone for the definition of cloud continuum that has evolved over time with the addition/removal of other keywords. It is also important to notice that no other grey literature works are mentioned by the selected studies.

Each of the 183 papers (181 from the initial search, and 2 from snowballing), was fully read by one of the authors independently and evaluated by the inclusion and exclusion criteria. As a result, we selected 36 papers.

[1]Scopus, https://www.scopus.com
[2]IEEEXplore Digital Library https://ieeexplore.ieee.org/
[3]ACM Digital Library: https://dl.acm.org
[4]Web of Science database: https://www.webofscience.com/

TABLE 3. Inclusion/Exclusion criteria.

Inc./Exc.	Criteria
Inclusion	Papers defining the concept of cloud continuum
Exclusion	Not in English Duplicated (post summarizing other websites) Out of topic (using the terms for other purposes) Non-peer-reviewed papers Work plans, roadmaps, vision papers, posters

C. DATA EXTRACTION STRATEGY

From the 36 Selected Papers (SP), we extracted the data that answers our research questions. Importantly, we extract the definitions on ''continuum'', the year of the publication, and the information on where the cloud is ''continued''. In addition to the key data mentioned above, we also extracted the type of publication (e.g. conference paper, or journal article).

TABLE 4. The information extracted from the selected papers.

RQ	Information Extracted	Motivation
RQ1	Definition of Continuum	
RQ2	Publication Year	To understand the chronological evolution of the definition.
RQ3	Cloud-to-* continuum	Identify the possible extensions of the cloud (*) mentioned in the SPs

The description of the information extracted, together with their motivation and the mapping to the RQs, is reported in Table 3.

D. KEYWORDING

The different definitions were written in natural language. Therefore, we needed to run a qualitative analysis among the authors, to identify similar definitions and different ones.

For this purpose, we applied a collective coding process to answer our RQs:

The manual identification of the aforementioned information was extracted collaboratively. From each paper, we first extracted the definition and print it to a post-it note (**RQ1**). Then, one author attached it to a whiteboard, and the other authors read all the other definitions proposed by the papers. All the authors discussed one by one the similarities and differences of each of the definitions, so as to decide whether to group them into a single definition or to create a new one.

Finally, the authors re-position the post-it notes reporting groups of similar definitions, and their key differences. For each definition, all the authors follow the same process to identify common aspects.

Last, authors highlighted with different colors the continuum extension to the cloud (**RQ3**)

IV. RESULTS

As expected, publications on Cloud Continuum are continuously growing in the recent years. The first definitions of cloud continuum were presented in [SP1] and [SP2] in 2016. For the following three years only four papers are identified

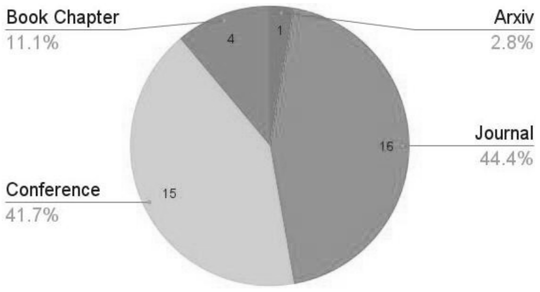


FIGURE 2. Selected papers by types.

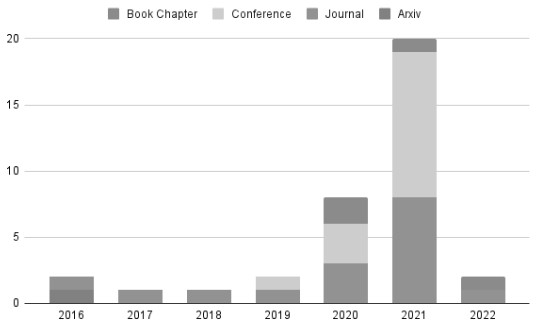


FIGURE 3. Selected Papers by Years.

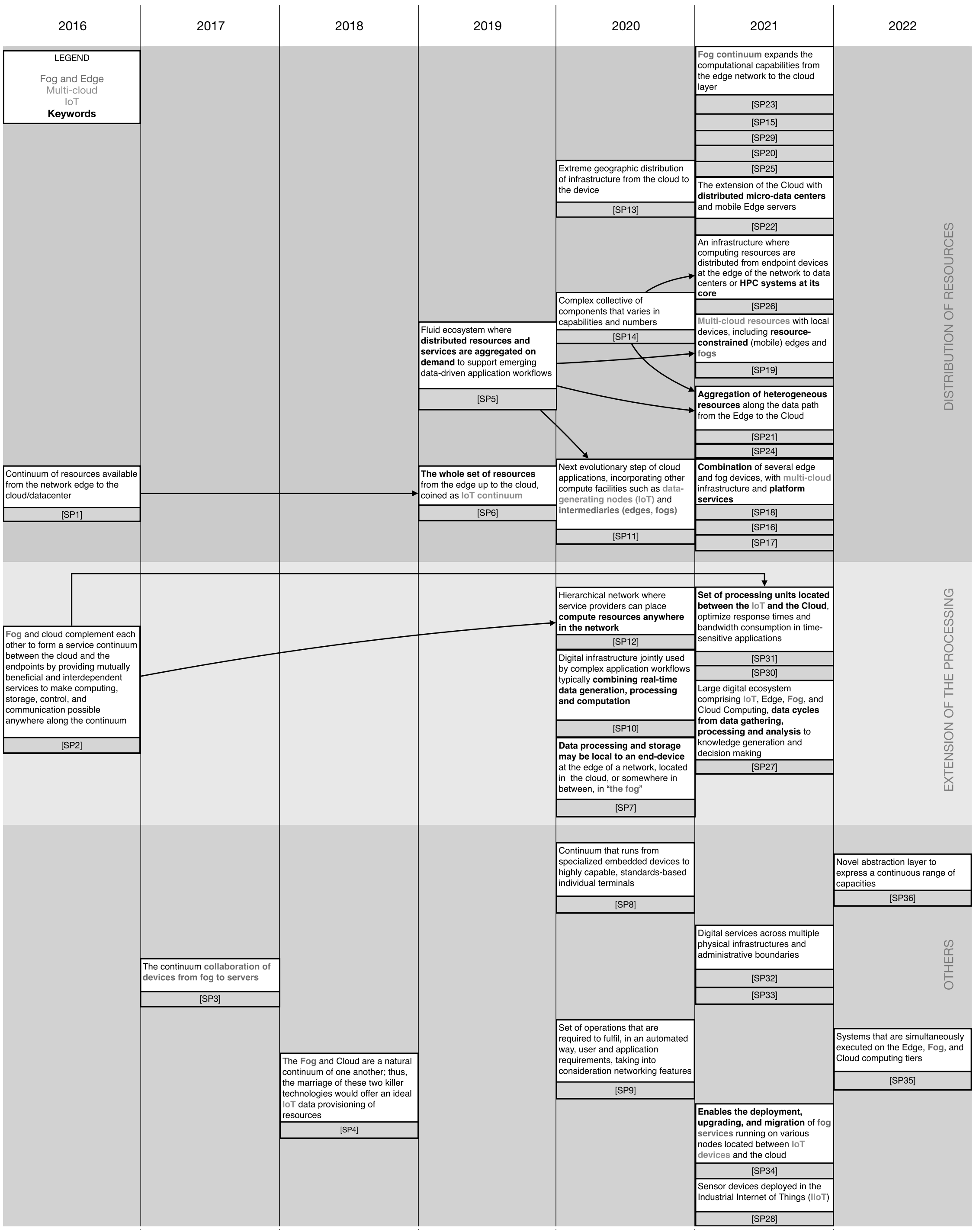
as related to the definition of cloud continuum. The interest in the topic started to grow in 2020. As depicted in Fig. 3 the majority of paper identified are from 2021. In the remainder of this Section, we answer our RQs.

A. THE DEFINITIONS OF CLOUD CONTINUUM (RQ1)

The first definitions of Cloud Continuum were both presented in 2016. Gupta et al. [SP1] defined cloud continuum as ''a continuum of resources available from the network edge to the cloud/datacenter'' while Chiang et al. [SP2] defined cloud continuum explicitly mentioning computational-related aspects, for instance, where and how the computation is performed.

We identify three main groups of definitions, with respect to their main aspects. Each group is represented by a block of a different colour in Fig. 4. The first and larger group contains all those sources defining cloud continuum as an aggregation/combination of different elements such as IoT devices, fog and edge nodes. In this case, cloud continuum only refers to the continuum of resources, but not of the computation. The second block contains all the sources defining cloud continuum with a particular focus on the processing/computation. Finally, we group together all those sources that do not belong to these two blocks.

Fig. 4 also shows that the definition of cloud continuum has two different origins. Both of the papers which gave origin to the definition, as presented previously, have been



Fog continuum expands the computational capabilities from the edge network to the cloud layer [SP23] [SP15] [SP29] [SP20] [SP25]

Extreme geographic distribution of infrastructure from the cloud to the device [SP13]

The extension of the Cloud with **distributed micro-data centers** and mobile Edge servers [SP22]

An infrastructure where computing resources are distributed from endpoint devices at the edge of the network to data centers or **HPC systems at its core** [SP26]

Complex collective of components that varies in capabilities and numbers [SP14]

Multi-cloud resources with local devices, including **resource-constrained** (mobile) edges and fogs [SP19]

Fluid ecosystem where **distributed resources and services are aggregated on demand** to support emerging data-driven application workflows [SP5]

Aggregation of heterogeneous resources along the data path from the Edge to the Cloud [SP21] [SP24]

Continuum of resources available from the network edge to the cloud/datacenter [SP1]

The whole set of resources from the edge up to the cloud, coined as IoT continuum [SP6]

Next evolutionary step of cloud applications, incorporating other compute facilities such as data-generating nodes (IoT) and **intermediaries (edges, fogs)** [SP11]

Combination of several edge and fog devices, with multi-cloud infrastructure and **platform services** [SP18] [SP16] [SP17]

Fog and cloud complement each other to form a service continuum between the cloud and the endpoints by providing mutually beneficial and interdependent services to make computing, storage, control, and communication possible anywhere along the continuum [SP2]

Hierarchical network where service providers can place **compute resources anywhere in the network** [SP12]

Set of processing units located between the IoT and the Cloud, optimize response times and bandwidth consumption in time-sensitive applications [SP31] [SP30]

Digital infrastructure jointly used by complex application workflows typically **combining real-time data generation, processing and computation** [SP10]

Large digital ecosystem comprising IoT, Edge, Fog, and Cloud Computing, **data cycles from data gathering, processing and analysis** to knowledge generation and decision making [SP27]

Data processing and storage may be local to an end-device at the edge of a network, located in the cloud, or somewhere in between, in "the fog" [SP7]

Continuum that runs from specialized embedded devices to highly capable, standards-based individual terminals [SP8]

Novel abstraction layer to express a continuous range of capacities [SP36]

Digital services across multiple physical infrastructures and administrative boundaries [SP32] [SP33]

The continuum **collaboration of devices from fog to servers** [SP3]

Set of operations that are required to fulfil, in an automated way, user and application requirements, taking into consideration networking features [SP9]

Systems that are simultaneously executed on the Edge, Fog, and Cloud computing tiers [SP35]

The Fog and Cloud are a natural continuum of one another; thus, the marriage of these two killer technologies would offer an ideal IoT data provisioning of resources [SP4]

Enables the deployment, upgrading, and migration of fog services running on various nodes located between IoT devices and the cloud [SP34]

Sensor devices deployed in the Industrial Internet of Things (IIoT) [SP28]

FIGURE 4. Definitions of cloud computing grouped by year and concepts. Each column represent a different year while the coloured blocks represents different aspects. Arrows between two blocks indicate that there is a direct citation to the definition.

published in 2016 but each of these focused on a different aspect. While the definition in [SP1] focused on the elements composing the system, the one proposed in [SP2] was centered around the concept of ''where happens what''.

The definition provided in [SP1] has been extended in 2019 from Kahvazadeh et al. [SP6] where the continuum of resources has been extended to 'the whole set of resources from the edge up to the cloud'. In parallel to this, Balouek-Thomert et al. [SP5], centered their definition on the concept of ''distributed resources services on demand''.

Within these groups we can identify some clusters. Each cluster combine multiple work within the same year defying the concept of cloud continuum in the same fashion. It is important to notice that each cluster is year-based as the definition evolved during the years (even when the author is the same). The highest amount of cluster can be found in the first group of work, those related to the distribution of resources.

Within this group we can find 3 different clusters. The first one includes 5 different work agreeing on the same definition which puts the concept of continuum strictly related to the concept of fog. The second cluster is composed of 3 works which stress the importance of having a combination of multiple edge and fog devices. The third cluster defines the cloud continuum as an aggregation of heterogeneous resources from the Edge to the cloud. The latter even tho it is composed of only two works, has a definition that focuses on the data path with a bottom-up design.

The other two clusters can be found one per each group. The first one, in the group ''extension of the processing'', includes two works defying cloud continuum as a Set of processing units located between the IoT and the Cloud. The other one, also including two works, focuses on the different services across multiple infrastructures.

B. THE EVOLUTION OF THE CLOUD CONTINUUM DEFINITION (RQ2)

In order to answer RQ2, we firstly extract the commonly adopted keywords of the cloud continuum definitions of the selected papers. Herein, based on the opinions of two domain experts, we extract six different keywords that delineate the characteristics (i.e., how, when and where) of cloud continuum and specify the entities (i.e., what) it connects. The keywords include:

- **Multi-Cloud:** definitions referring to multiple cloud entities;
- **Fog:** definitions explicitly referring to Fog;
- **IoT:** definitions referring to internet of things, IoT, things;
- **Anywhere:** definitions explicitly reporting that the computation can be executed everywhere;
- **Micro Datacenters:** definitions explicitly reporting the use of micro datacenters to the goal of providing low-latency access to data processing and data storage.
- **Simultaneous:** definitions explicitly reporting that the computation can be simultaneously executed on multiple nodes.

Therefore, by summarizing the adoptions of these keywords by the selected papers in chronological order (reported in TABLE 5), we can observe the evolution of the cloud continuum definition.

The two earliest definitions, [SP1] and [SP2] in 2016, both anchored the concept of fog between cloud and edge, where the term ''continuum'' was firstly used by its literal meaning in this context. Specially, Chiang and Zhang [SP2] emphasized that within such continuum, services like computing, storage, control and communication could be provided anywhere between cloud and edge. From 2017 to 2018, the two studies, [SP3] and [SP4] continued adopting the term ''continuum'' describing the combination of fog and cloud, when Peng et al. [SP4] indicated that the continuum of fog and cloud could provide ideal IoT data provisioning. In 2019, Balouek-Thomert et al. [SP5] also mentioned ''computing continuum'' as a fluid ecosystem with aggregated resources and services but didn't emphasized its positioning between cloud and edge. Meanwhile, also in 2019, Kahvazadeh et al. [SP6] proposed the term ''IoT continuum'' but similarly coined the definition as a whole set of resources between edge and cloud.

Since 2020, the number of studies that provided definitions to cloud continuum has been growing sharply. In 2020, eight studies mentioned the concept of ''continuum'' and similarly placed the concept as the services between cloud and the end-devices (i.e., edge). However, though five studies, [SP7], [SP8], [SP11], [SP12], and [SP14], mentioned ''fog'' when defining continuum, none of the studies have clearly distinguish them; when some studies, e.g., [SP8], [SP12], indicate continuum is between cloud and fog. Meanwhile, four studies mentioned IoT when defining continuum [SP11]-[SP14]; however, the relation between continuum and IoT is not clearly delineate either. On the other hand, Kassir et al. [SP12] also indicate that compute resources can be placed anywhere in the network when citing [SP2]. Furthermore, Spillner et al. [SP11] emphasize that continuum is more than simply a ''multi-cloud'' but incorporating other compute facilities, e.g., mobile devices, IoT sensor nodes, edges and fogs, which is the first time continuum is connected with the notion of ''multi-cloud''.

In 2021, nine studies mentioned ''fog'' as a critical entity in the definition of continuum. Different from previously, many of these studies, e.g., [SP18]-[SP20], [SP22], [SP23], have anchored the continuum concept as the combination or aggregation of several fog, edge, IoT devices or services, or the extension of the cloud. Meanwhile, four studies [SP16]-[SP19] also indicate that cloud continuum is a ''multi-cloud'' infrastructure. On the other hand, eight studies indicate that IoT is a crucial part of the cloud continuum concept when, however, the interpretation of the term differs slightly. For example, Xhafa and Krause [SP27] define cloud continuum as a large digital ecosystem comprising IoT, Edge, Fog, and

TABLE 5. Initial search results by sources.

Reference	Year	Architecture				Performance	
		Multi-Cloud	Fog	IoT	Micro Datacenters	Anywhere	Simultaneous
[SP1]	2016		✓				
[SP2]	2016		✓			✓	
[SP3]	2017		✓				
[SP4]	2018		✓	✓			
[SP5]	2019						
[SP6]	2019			✓			
[SP7]	2020		✓				
[SP8]	2020		✓				
[SP9]	2020						
[SP10]	2020						
[SP11]	2020	✓	✓	✓			
[SP12]	2020		✓	✓		✓	
[SP13]	2020			✓			
[SP14]	2020		✓	✓			
[SP15]	2021						
[SP16]	2021	✓					
[SP17]	2021	✓					✓
[SP18]	2021	✓	✓				
[SP19]	2021	✓	✓				
[SP20]	2021		✓				
[SP21]	2021						
[SP22]	2021		✓		✓		
[SP23]	2021		✓				
[SP24]	2021						
[SP25]	2021						
[SP26]	2021						
[SP27]	2021		✓	✓			
[SP28]	2021			✓			
[SP29]	2021			✓			
[SP30]	2021		✓	✓			
[SP31]	2021		✓	✓			
[SP32]	2021			✓			
[SP33]	2021			✓			
[SP34]	2021		✓	✓			
[SP35]	2022		✓				✓
[SP36]	2022						

etc., where IoT is the individual entity/device providing services; Zeiner and Unterberger [SP28] defines edge-to-cloud continuum as *a data-driven Internet of Things combines the physical world with the world of information*, where IoT is referred to as the assembly instead of the individual. Specially, Mehran et al. [SP22] define cloud continuum as the extension of the cloud with distributed micro-datacenters and mobile edge servers, which is the first and only time when micro-datacenter is used.

Until February 2022, two studies also provided definitions to cloud continuum. Dustdar et al. [SP35] define it by emphasizing it is the system that is ''simultanously'' executed on the edge, fog, and cloud computing tiers. Similarly, in 2021, Risco et al. [SP17] also mentioned the term ''simultanously'' by indicating cloud continuum ''*simultaneously involves both on-premises and public Cloud platforms to process data captured at the edge*''. The other definition given by Spillner et al. did not specify the entities that cloud continuum aggregating but emphasize it is an ''*novel abstraction layer to express a continuous range of capacities*''.

C. WHERE DOES THE CLOUD CONTINUE (RQ3)

Among the 36 SPs, nine of them mention the continuum as ''cloud-to-thing(s) continuum''. Therein, these studies indicate that cloud continuum connects or is placed between cloud(s) and the IoT-connected devices (i.e., things). Specially, Kassir et al. [SP12] state that ''cloud-to-thing(s) continuum'' is equivalent to ''Fog-to-Cloud continuum''. Meanwhile, two studies, [SP22] and [SP23], use ''Cloud-fog continuum'' or ''fog continuum'' indicating the continuum extends the cloud towards fog, which could either refer to fog nodes (i.e., also things) or fog in general.

On the other hand, seven papers amongst the 36 SPs use the term ''Edge-to-Cloud continuum'' (or Cloud-edge continuum, or edge/cloud continuum) indicating the cloud ''continues'' towards edge nodes. Kahvazadeh et al. [SP6] use the term ''IoT continuum'' but describe the same connection between cloud and edge. Three studies use directly the term ''cloud continuum'' but also define it as combination of cloud and edge.

Furthermore, ten studies use ''Computing Continuum'' to emphasize the computing capability instead of the connection of entities. Within these definitions, the ''continuum'' can be used connecting any entities, e.g., edge, fog, local devices (i.e., IoT or things), data centers, etc. Specially, Balouek-Thomert et al. [SP5] do not describe the specific nodes being connected by continuum but defines ''computing continuum'' as ''a digital infrastructure jointly used by complex application workflows''. Beckman et al. [SP14] provide a similar definition as ''a collective of components with various capabilities and numbers in aggregate''. Spillner et al. [SP36] provide a high-level abstracted definition of computing

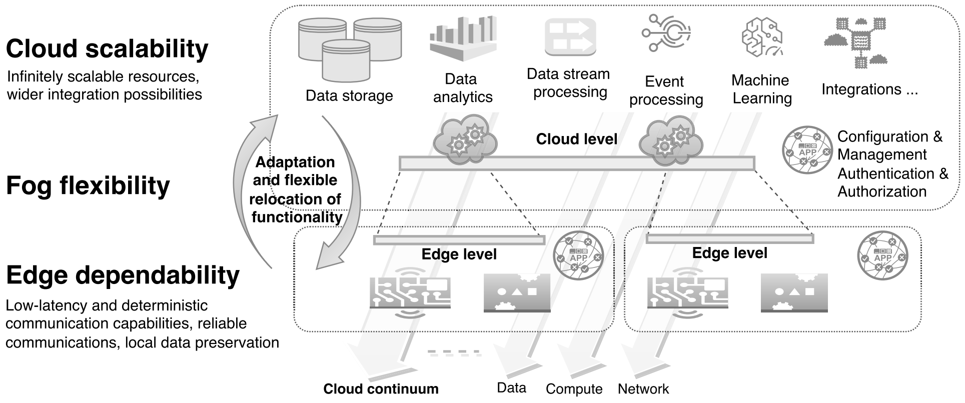

Cloud scalability
Infinitely scalable resources, wider integration possibilities
Fog flexibility
Edge dependability
Low-latency and deterministic communication capabilities, reliable communications, local data preservation
Data storage
Data analytics
Data stream processing
Event processing
Machine Learning
Integrations ...
Cloud level
Configuration & Management Authentication & Authorization
Adaptation and flexible relocation of functionality
Edge level
Edge level
APP
Cloud continuum
Data
Compute
Network

FIGURE 5. Architecture of Cloud Continuum.

continuum as ''novel abstraction layer to express a continuous range of capacities''

Comparatively, early studies, e.g., [SP1] and [SP2], did not try to provide distinguishable terms but only use the term ''continuum'' literally trying to describe the conceptual idea. Similarly, these two studies also place the ''continuum'' between cloud to edge or cloud to fog.

V. DISCUSSION

Several definitions of Cloud Continuum have been proposed in the last six years. However, only few have been used or extended.

It is interesting to notice the two main types of definitions, one considering the continuum as distribution of resources in different network elements, including IoT, Fog, Edge, but also HPC, while the other definition considering the continuum as an extension of the processing power to different nodes, often mentioning the possibility of executing also AI.

The investigation of the different cloud continuum concepts allowed us to draw an overall architecture of the cloud continuum (Figure 5)

Based on the analysis conducted in this work, we can propose a new definition of cloud continuum, combining the most frequently mentioned aspects.

Cloud Continuum is an extension of the traditional Cloud towards multiple entities (e.g., Edge, Fog, IoT) that provide analysis, processing, storage, and data generation capabilities.

A. FUTURE CHALLENGES

The results of this work enabled us to distill a set of challenges for the cloud continuum. Therein, the majority of the SPs point out the challenges concerning the **dynamic allocation of the computation**([SP31], [SP22]), and in particular of the execution of the AI, and the related resource orchestration, network partitioning ([SP30]) and support for context-awareness ([SP9]).

As part of the resource orchestration, **job scheduling** is also identified as one of the most common challenges that need to be addressed in the future ([SP4], [SP13], [SP14], [SP16], [SP24]). Tools such as Kafka-ML ([SP30]) and network virtualization ([SP1]) are proposed towards such an end. Furthermore, other techniques, e.g., adopting APIs ([SP16]) and game theory ([SP20]), are proposed as promising solutions for application deployment and orchestration.

The **robustness** of the cloud continuum is also considered a critical aspect for the future. For example, [SP11] highlights the complexity of the awareness of application deployment towards the adaptation for higher resilience. [SP23] and [SP15] also indicate that tolerant IoT services and **self-healing** components shall serve for the future steps towards structural and behavioral optimization of cloud continuum system.

Furthermore, **security** of the cloud continuum systems ([SP2], [SP4]) is also a key aspect when specific techniques, e.g., Information-Centric Network integration ([SP9]), and Hybrid key distribution ([SP6]) are seen as future works.

Other **performance** characteristics, e.g., scalability ([SP28]), mobility ([SP23]) and consistency ([SP2]), together with the corresponding ways of acquisition ([SP11]), comparison ([SP29]) and benchmark ([SP26]) are also mentioned as the challenges.

Meanwhile, other future challenges include high-level abstraction models and architectural trade-off ([SP2], [SP10]), [SP14]), interfaces and user experience ([SP1], [SP2]), positioning and localization, ([SP4]) and the Incentives of device participation ([SP2]). The researchers shall consider contributing to the solutions to the above challenges in order to enrich the domain knowledge of cloud continuum research.

B. THREATS TO VALIDITY

We are aware that our work is subject to threats to validity. The terms Cloud, Fog, IoT, and Edge are sufficiently stable to be used as search strings. In order to assure the retrieval of all papers on the selected topic, we searched broadly in general publication databases, which index most well-reputed publications. To improve the reliability of this work, we defined search terms and applied procedures that can be replicated by others. Since this is a mapping study and no systematic review, the inclusion/exclusion criteria are only related to whether the topic of Cloud Continuum is present in a paper or not, as suggested by [20]. As for the analysis procedure, since our analysis only uses descriptive statistics, the threats are minimal. However, we are aware that the synthesis of the definition might be subjective. To mitigate this threat, the analysis was done collaboratively, using a collecting coding methods, and discussing with all the authors about inconsistencies. The Kohen K index about our disagreement also confirms the quality of the qualitative analysis performed.

VI. CONCLUSION

In this work, we proposed a systematic mapping study on the definition of Cloud Continuum to obtain an overview of its existing definitions and how the concept has been evolved.

We identified 36 studies which proposed definitions to Cloud Continuum dated from 2016. All these definitions are summarized in Figure 4. We organized all the 36 existing definitions in chronological order.

In conclusion, we propose to complement existing definitions into a common one that merges explicitly two aspects: the continuum as **extension of the resources**, and as **extension of computational capabilities**.

As a result, we formulated the definition of cloud continuum as "**an extension of the traditional Cloud towards multiple entities (e.g., Edge, Fog, IoT) that provide analysis, processing, storage, and data generation capabilities.**"

The new definition enables both practitioners and researchers to better understand the concept of cloud continuum and to gain insights into the potential advance in service-oriented computing.

As regards future work, we are planning to extend this work in the context of cognitive continuum.

APPENDIX A: THE SELECTED PAPERS

[SP1] Gupta, H. et al., 2016. SDFog: A software defined computing architecture for QoS aware service orchestration over edge devices. arXiv preprint arXiv:1609.01190.

[SP2] Chiang, M. and Zhang, T., 2016. Fog and IoT: An overview of research opportunities. IEEE Internet of things journal, 3(6), pp.854-864.

[SP3] Coughlin, T., 2017. Convergence through the cloud-to-thing consortium [future directions]. IEEE Consumer Electronics Magazine, 6(3), pp.14-17.

[SP4] Peng, L. et al., 2018. Toward integrated Cloud–Fog networks for efficient IoT provisioning: Key challenges and solutions. Future Generation Computer Systems, 88, pp.606-613.

[SP5] Balouek-Thomert, D. et al., 2019. Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows. The International Journal of High Performance Computing Applications, 33(6), pp.1159-1174.

[SP6] Kahvazadeh, S. et al., 2019. Securing combined fog-to-cloud systems: challenges and directions. In FTC 2019 (pp. 877-892). Springer.

[SP7] Domaschka, J. et al., 2020. Towards an architecture for reliable capacity provisioning for distributed clouds. In Managing Distributed Cloud Applications and Infrastructure (pp. 1-25). Palgrave Macmillan, Cham.

[SP8] Milojicic, D., 2020. The edge-to-cloud continuum. Computer, 53(11), pp.16-25.

[SP9] Da Silva, D.M.A. and Sofia, R.C., 2020. A discussion on context-awareness to better support the IoT cloud/edge continuum. IEEE Access, 8, 193686-193694.

[SP10] Rosendo, D. et al., 2020. E2clab: Exploring the computing continuum through repeatable, replicable and reproducible edge-to-cloud experiments. In 2020 CLUSTER (pp. 176-186). IEEE.

[SP11] Spillner, J. et al., 2020. Rule-based resource matchmaking for composite application deployments across IoT-fog-cloud continuums. In 2020 IEEE/ACM UCC (pp. 336-341). IEEE.

[SP12] Kassir, S. et al., 2020. Service placement for real-time applications: Rate-adaptation and load-balancing at the network edge. In CSCloud 2020/EdgeCom 2020 (pp. 207-215). IEEE.

[SP13] Bendechache, M. et al., 2020. Simulating resource management across the cloud-to-thing continuum: A survey and future directions. Future Internet Journal, 12(6), p.95.

[SP14] Beckman, P. et al., 2020. Harnessing the computing continuum for programming our world. Fog Computing: Theory and Practice, pp.215-230.

[SP15] Alonso, J. et al., 2021. Optimization and Prediction Techniques for Self-Healing and Self-Learning Applications in a Trustworthy Cloud Continuum. Information, 12(8), p.308.

[SP16] Luckow, A. et al., 2021. Exploring task placement for edge-to-cloud applications using emulation. In 2021 ICFEC (pp. 79-83). IEEE.

[SP17] Risco, S. et al., 2021. Serverless workflows for containerised applications in the cloud continuum. Journal of Grid Computing, 19(3), pp.1-18.

[SP18] Spillner, J., 2021. Self-balancing architectures based on liquid functions across computing continuums. In UCC (pp. 1-6).

[SP19] Hass, D. and Spillner, J., 2021. Interactive application deployment planning for heterogeneous computing continuums. In AINA (pp. 551-560). Springer, Cham.

[SP20] Kimovski, D. et al., 2021. Cloud, Fog, or Edge: Where to Compute?. IEEE Internet Computing, 25(4), pp.30-36.

[SP21] Balouek-Thomert, D. et al., 2021. Evaluating policy-driven adaptation on the Edge-to-Cloud Continuum. In 2021 UrgentHPC (pp. 11-20). IEEE.

[SP22] Mehran, N. et al., 2021. A Two-Sided Matching Model for Data Stream Processing in the Cloud–Fog Continuum. In 2021 CCGrid (pp. 514-524). IEEE.

[SP23] Nezami, Z. et al., 2021. Decentralized edge-to-cloud load balancing: Service placement for the Internet of Things. IEEE Access, 9, pp.64983-65000.

[SP24] Balouek-Thomert, D. et al., 2021. MDSC: modelling distributed stream processing across the edge-to-cloud continuum. In UCC 2021 (pp. 1-6).

[SP25] Kimovski, D. et al., 2021. Mobility-Aware IoT Applications Placement in the Cloud Edge Continuum. IEEE Transactions on Services Computing.

[SP26] Rosendo, D. et al., 2021. Reproducible performance optimization of complex applications on the edge-to-cloud continuum. In CLUSTER (pp. 23-34). IEEE.

[SP27] Xhafa, F. and Krause, P., 2021. IoT-Based Computational Modeling for Next Generation Agro-Ecosystems: Research Issues, Emerging Trends and Challenges. In IoT-based Intelligent Modelling for Environmental and Ecological Engineering (pp. 1-21). Springer, Cham.

[SP28] Zeiner, H. and Unterberger, R., 2021. Time-aware Data Spaces-A key Computing Unit in the Edge-to-Cloud Continuum. In 2021 FiCloud (pp. 250-255). IEEE.

[SP29] Dizdarević, J. and Jukan, A., 2021. Experimental Benchmarking of HTTP/QUIC Protocol in IoT Cloud/Edge Continuum. In ICC 2021 (pp. 1-6). IEEE.

[SP30] Carnero, A. et al., 2021. Managing and Deploying Distributed and Deep Neural Models Through Kafka-ML in the Cloud-to-Things Continuum. IEEE Access, 9, pp.125478-125495.

[SP31] Torres, D.R. et al., 2021. An open source framework based on Kafka-ML for Distributed DNN inference over the Cloud-to-Things continuum. Journal of Systems Architecture, 118, p.102214.

[SP32] Alberternst, S. et al., 2021. From Things into Clouds–and back. In 2021 CCGrid (pp. 668-675). IEEE.

[SP33] Alberternst, S. et al., 2021. Orchestrating Heterogeneous Devices and AI Services as Virtual Sensors for Secure Cloud-Based IoT Applications. Sensors, 21(22), p.7509.

[SP34] Čilić, I. et al., 2021. Towards Service Orchestration for the Cloud-to-Thing Continuum. In 2021 SpliTech (pp. 01-07). IEEE.

[SP35] Dustdar, S. et al., 2022. On distributed computing continuum systems. IEEE Transactions on Knowledge and Data Engineering.

[SP36] Spillner, J. et al., 2022. Intent-Based Placement of Microservices in Computing Continuums. Future Intent-Based Networking, 38-50

REFERENCES

[1] (2020). *Cloud Computing: Towards a Smart Cloud Computing Continuum*. Accessed: Jul. 7, 2022. [Online]. Available: https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/ict-40-2020

[2] (2021). *Cloud Computing and High-Throughput Computing Resources for Collaborative Research in Computational Neuroscience (CRCNS) Grantees*. Accessed: Jul. 7, 2022. [Online]. Available: https://beta.nsf.gov/funding/opportunities/cloud-computing-and-high-throughput-computing-resources-collaborative

[3] (2022). *Cognitive Cloud: AI-Enabled Computing Continuum From Cloud to Edge (RIA)*. Accessed: Jul. 7, 2022. [Online]. Available: https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/horizon-cl4-2022-data-01-02

[4] M. A. Al-Sharafi, R. A. Arshah, and E. A. Abu-Shanab, "Factors influencing the continuous use of cloud computing services in organization level," in *Proc. Int. Conf. Adv. Image Process.*, Aug. 2017, pp. 189–194.

[5] M. Asim, Y. Wang, K. Wang, and P. Q. Huang, "A review on computational intelligence techniques in cloud and edge computing," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 4, no. 6, pp. 742–763, Dec. 2020.

[6] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless edge computing: Vision and challenges," in *Proc. Australas. Comput. Sci. Week Multiconf.*, New York, NY, USA, Feb. 2021, pp. 1–10, doi: 10.1145/3437378.3444367.

[7] M. Bendechache, S. Svorobej, P. T. Endo, and T. Lynn, "Simulating resource management across the cloud-to-thing continuum: A survey and future directions," *Future Internet*, vol. 12, no. 6, p. 95, May 2020.

[8] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The Internet of Things, fog and cloud continuum: Integration and challenges," *Internet Things*, vols. 3–4, pp. 134–155, Oct. 2018.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[10] M. Chiang, S. Ha, F. Risso, T. Zhang, and I. Chih-Lin, "Clarifying fog computing and networking: 10 questions and answers," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 18–20, Apr. 2017, doi: 10.1109/MCOM.2017.7901470.

[11] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Edge computing meets millimeter-wave enabled VR: Paving the way to cutting the cord," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6, doi: 10.1109/WCNC.2018.8377419.

[12] K. E. Emam, "Benchmarking Kappa: Interrater agreement in software process assessments," *Empirical Softw. Eng.*, vol. 4, no. 2, pp. 113–133, 1999.

[13] M. Ghobaei-Arani, A. Souri, and A. A. Rahmanian, "Resource management approaches in fog computing: A comprehensive review," *J. Grid Comput.*, vol. 18, no. 1, pp. 1–42, Mar. 2020.

[14] J. Kampars, D. Tropins, and R. Matisons, "A review of application layer communication protocols for the IoT edge cloud continuum," in *Proc. 62nd Int. Sci. Conf. Inf. Technol. Manage. Sci. Riga Tech. Univ. (ITMS)*, Oct. 2021, pp. 1–6.

[15] P. Kansal, M. Kumar, and O. P. Verma, "Classification of resource management approaches in fog/edge paradigm and future research prospects: A systematic review," *J. Supercomput.*, vol. 2022, pp. 1–60, Mar. 2022.

[16] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," School Comput. Sci. Math., Softw. Eng. Group, Keele Univ., Keele, U.K., EBSE Tech. Rep. EBSE-2007-01, Version 2.3, 2007.

[17] P. Nguyen, N. Ferry, G. Erdogan, H. Song, S. Lavirotte, J.-Y. Tigli, and A. Solberg, "Advances in deployment and orchestration approaches for IoT—A systematic review," in *Proc. IEEE Int. Congr. Internet Things (ICIOT)*, Jul. 2019, pp. 53–60.

[18] J. Nupponen and D. Taibi, "Serverless: What it is, what to do and what not to do," in *Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, Mar. 2020, pp. 49–50, doi: 10.1109/ICSA-C50368.2020.00016.

[19] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, ''Cloud container technologies: A state-of-the-art review,'' *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 677–692, Jul. 2019.

[20] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, ''Systematic mapping studies in software engineering,'' in *Proc. Electron. Workshops Comput.*, Jun. 2008, pp. 1–10.

[21] S. Ramanathan, N. Shivaraman, S. Suryasekaran, A. Easwaran, E. Borde, and S. Steinhorst, ''A survey on time-sensitive resource allocation in the cloud continuum,'' *Inf. Technol.*, vol. 62, nos. 5–6, pp. 241–255, Dec. 2020.

[22] A. Spataru, ''A review of blockchain-enabled fog computing in the cloud continuum context,'' *Scalable Comput., Pract. Exper.*, vol. 22, no. 4, pp. 463–468, Dec. 2021.

[23] S. Svorobej, M. Bendechache, F. Griesinger, and J. Domaschka, ''Orchestration from the cloud to the edge,'' in *The Cloud-to-Thing Continuum*. Cham, Switzerland: Springer, 2020, pp. 61–77.

[24] Y. Wei and M. B. Blake, ''Service-oriented computing and cloud computing: Challenges and opportunities,'' *IEEE Internet Comput.*, vol. 14, no. 6, pp. 72–75, Nov. 2010.

[25] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, ''All one needs to know about fog computing and related edge computing paradigms: A complete survey,'' *J. Syst. Archit.*, vol. 98, pp. 289–330, Sep. 2019, doi: 10.1016/j.sysarc.2019.02.009.

SERGIO MORESCHINI (Student Member, IEEE) is currently pursuing the Ph.D. degree with the Faculty of Information Technology and Communication Sciences, Tampere University, Finland. He is also a Researcher at the Cloud Software Evolution and Assessment (CloudSEA) Research Group. His research interest includes tools for MLOps. He also contributes actively to the domains of empirical software engineering, open-source software quality, and data-driven software engineering.

FABIANO PECORELLI (Associate Member, IEEE) received the bachelor's, master's, and Ph.D. degrees in computer science from the University of Salerno, Italy. He is currently a Postdoctoral Researcher with the Cloud Software Evolution and Assessment (CloudSEA) Research Group, Tampere University, Finland. His research interests include software code and test quality, predictive analytics, mining software repositories, software maintenance and evolution, and empirical software engineering. He serves and had served as a referee for various international journals in the field of software engineering (e.g., IEEE Transactions on Software Engineering, IEEE Transactions on Software Engineering, *EMSE*, and *JSS*).

XIAOZHOU LI (Associate Member, IEEE) is currently a Postdoctoral Researcher with the Faculty of Information Technology and Communication Sciences, Tampere University, Finland. He is also a Researcher at the Cloud Software Evolution and Assessment (CloudSEA) Research Group. His research interests include open-source software quality, software maintenance and evolution, user review opinion mining, data-driven empirical software engineering, computational game studies, and gamification design.

SONIA NAZ received the master's degree in computer science from Fatima Jinnah Women's University, Pakistan, and the master's degree in software engineering from International Islamic University, Pakistan. She is currently pursuing the Ph.D. degree with the Faculty of Information Technology and Communication Sciences, Tampere University, Finland. She is a part of the Cloud Software Evolution and Assessment (CloudSEA) Research Group. Her research area is empirical software engineering with a particular focus on agile processes for software development. She was a Lecturer in the computer science course with the University of Wah, Pakistan, from 2015 to 2020.

DAVID HÄSTBACKA (Member, IEEE) received the D.Sc.(Tech.) (Hons.) and M.Sc. (Tech.) degrees from the Tampere University of Technology, in 2013 and 2007, respectively. He is currently an Assistant Professor (Tenure Track) at Tampere University, Finland. His research interests include system and software architectures, and interoperability of software systems in production and energy systems applications.

DAVIDE TAIBI (Member, IEEE) is currently a Full Professor at the University of Oulu, Finland, where he heads the M3S Cloud Research Group. His research is mainly focused on empirical software engineering applied to cloud native systems, with a special focus on the migration from monolithic to cloud-native applications. He is investigating processes, and techniques for developing cloud native applications, identifying cloud-native specific patterns, and anti-patterns. He has been a member of the International Software Engineering Network (ISERN), since 2018. Before moving to Finland, he was an Assistant Professor at the Free University of Bozen/Bolzano (2015–2017), a Postdoctoral Research Fellow at the Technical University of Kaiserslautern and Fraunhofer Institute for Experimental Software Engineering (IESE) (2013–2014), and a Research Fellow at the University of Insubria (2007–2011).

PUBLICATION IV

Smart Edge Service Update Scheduler: An Industrial Use Case

Sergio Moreschini, Francesco Lomio, David Hästbacka, and Davide Taibi

In: *Service-Oriented Computing – ICSOC 2022 Workshops*. Springer Nature Switzerland, 2023, pp. 171–182
DOI: https://doi.org/10.1007/978-3-031-26507-5_14

Smart Edge Service Update Scheduler: An Industrial Use Case

Sergio Moreschini[1], Francesco Lomio[1], David Hästbacka[1], and Davide Taibi[1,2]

[1] Tampere University, Tampere, Finland {name.surname}@tuni.fi
[2] University of Oulu, Oulu, Finland {name.surname}@oulu.fi

Abstract. Software systems need to be maintained and frequently updated to provide the best possible service to the end-users. However, updates sometimes, cause the system or part of it to restart and disconnect, causing downtime and potentially reducing the quality of service. In this work we studied and analyzed the case of a large Nordic company running a service-oriented system running on edge nodes, and providing services to 270K IoT devices. To update the system while minimizing downtime, we develop a smart edge service update scheduler for a service-oriented architecture, which suggests the best possible update schedule that minimizes the loss of connections for IoT devices.
Our approach was validated by applying the scheduling algorithm to the whole system counting 270k edge nodes distributed among 800 locations. By taking into account the topology of the software system and its real-time utilization, it is possible to optimize the updates in a way that substantially minimizes downtime.

Keywords: Edge computing · provisioning · update scheduling · service-oriented · IoT.

1 Introduction

Software systems constantly need to be updated. Modern agile methods allow the continuous development and deployment of changes. However, the deployment of updates can require the restart of the system or part of it.

When considering widely used software systems, and in particular critical systems, downtime is usually unacceptable, and different strategies should be considered to avoid or minimize downtime as much as possible.

In our case, a very large Nordic company[3] is running a service-based system. The system is running 24/7 and it is deployed on edge and cloud, in multiple countries. Among different countries, the system has more than 800 locations, with an average of 336 edge nodes for each location for a total of $\sim 270k$ edge nodes. Each edge node provides a service to 100-1000 users connected simultaneously totaling 270 million connected Internet of Things (IoT) devices.

[3] For reasons of NDA, we are not allowed to disclose the name of the company, nor the low-level details of the use case.

The company is continuously developing the system using an agile methodology and the continuous building of the system needs to deploy a new version of the code every day. However, the deployment of the new version requires the restart of each location, taking an average of 30 minutes, and impacting all the edge nodes and related services provided to the connected IoT. During this time, all the end users connected to the edge nodes in the location, need to be rerouted to another edge node in a different location, to minimize the number of dropped service calls. However, the IoT devices can access only adjacent locations, due to the wireless technology adopted, increasing the complexity of the updates.

Given the daily upgrade time frame and the number of nodes, it would not be feasible to have a sequential upgrade schedule, as it would require more than 405 hours ($\sim$ 16 days).

Therefore, we intervened to support the company in identifying a smart update algorithm to schedule the updates of each location while reducing the number of service call drops as much as possible, maximizing the quality of service.

For this purpose, we defined a smart scheduling algorithm, validated it, and finally deployed it in production. The goal of the scheduling systems is to provide the suggested timing at which each location should start the provisioning process.

As a result, the company is now able to continuously deploy new updates, dropping only once a day, for 30 minutes, 20% of the calls to the service APIs.

The result of this work can be useful to researchers to validate the scheduling algorithm and to further extend it. Moreover, companies can benefit from this work by applying and extending it in production. It is important to remember that this algorithm is currently deployed in production, on a very large-scale system.

The remainder of this paper is structured as follows. In the next section, we introduce the necessary background and related work. In section 3 we introduce the smart edge service update scheduler, explaining its characteristics and its rationale. In section 4 we describe how the performance of the scheduling algorithm is measured. Section 5 includes the validation of the algorithm and the smart edge provisioning scenario. Section 6 finally presents our conclusions and draws future works.

2 Background and Related Works

2.1 Provisioning

The term provisioning is usually referred to as the process of preparing and equipping a software system to provide the best possible services to its users. However, since a system needs to be updated constantly, a vital part of the provisioning process is related to the updates and eventual restart of the applications. In this work, we use the term provisioning exactly to describe the update process of edge nodes (with consequent rebooting) to provide the best QoS to the system's users.

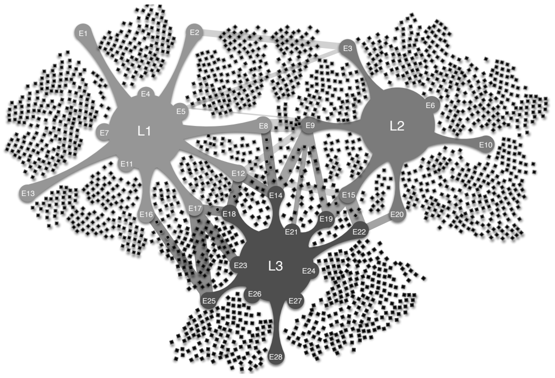


Fig. 1. Example of a system with multiple *locations* (L1, L2, and L3), each with multiple *edge nodes* (E1, E2, ..), and a variety of *IoT devices* connected to the the edge nodes of the closest location (squares with same colors as locations). Moreover, the lines connecting edge nodes of different locations indicate the possibility to handover the connections of the IoT devices.

2.2 Edge Computing

With the term Edge computing, we refer to a system where the computation is brought closer to the end user and the source of the data [1]. By keeping the majority of the data closer to the end users, there is a significant advantage in terms of lower latency and improved bandwidth compared to centralized systems. For this reason, whenever real-time processing is needed, edge computing allows bringing computation and data storage closer to the client.

2.3 Related Works

The increase in usage of edge technology and computing, including the proliferation of IoT devices, has increased the need for additional care needed to guarantee a sufficient quality of service. This system decentralizes the use of computational resources, bringing new issues in the management of the overall network.

Most of the research has therefore focused on how to optimize the provisioning of the resources for edge systems. Kherraf et al. [2], for example, proposed an approach that decomposes the resource provisioning and workload assignment into subtasks, allowing for higher performance trends in the overall system. Similarly, Cai et al [3] proposed a provisioning model called *edge federation*, which allows to schedule of resources among multiple edge infrastructure providers by characterizing the provisioning as a linear programming problem. Their method resulted in significantly reduced costs. Xu et al. [4] also tried to optimize the provisioning of resources in edge computing, by proposing a dynamic provisioning method which, besides optimizing the resource scheduling, also tries to optimize energy consumption and the completion time. Another issue in resource provisioning is that sometimes it doesn't take into account edge-specific characteristics. Ogbuachi et al. [5] tackled this problem by integrating real-time information regarding physical, operational, and network parameters in the scheduling of 5G edge computing, showing that this approach improves the scheduling process compared to the default Kubernetes scheduler.

Among the works that tackled the resource provisioning process, some of them exploited machine learning models for optimizing it. Guo et al. [6], for instance, used a combination of Auto-Regressive Integrated Moving Average (ARIMA) and Back-Propagation Neural Network (BPNN), to predict the load and optimize the resource provisioning of an edge system. Similarly, Li et al. [7] used the same ARIMA and BPNN models to forecast the load and proposed a location for new requests to be filled, reducing the cost of provisioning.

3 The Scheduling Algorithm

The proposed smart scheduling algorithm is based on three different contribution factors, as described below.

– **Static Weight**: a factor relates to the information which is not going to change in the near future and, therefore, *static* in time. It is computed taking into account the topology of the system.
– **Dynamic Weight**: this factor, in contrast, includes all of the information which is not static in time and therefore related to throughput among different nodes. Specifically, in this model, the Dynamic weight is related to the number of active connections each location has at different time slots.
– **Cluster ID**: this factor assigns a value to each location showing which are similar and can be considered in the same cluster.

3.1 Static Weight

The topology of the network is presented as a structured file including the Edge Source and the Destination Edge for each different possible service. This means that between different couple of locations it is possible to have multiple edge-based connections. Moreover, we also know that different locations have a different number of edge nodes. As we believe that different information has different importance when impacting the topology of the network we computed the static weight s_w of each location as a weighted sum of this different information so that :

$$s_w = \alpha * W_E + \beta * W_L, \tag{1}$$

where α and β are two factors assigned to the different weights, W_E is the weight computed based on the number of edges within a single location and W_L is the weight computed based on how many connections there are between two different locations.

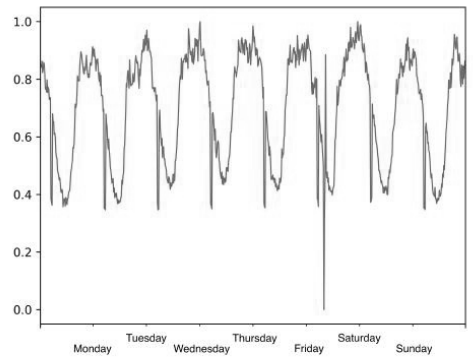


Fig. 2. Seasonality of the active traffic for a specific location.

More in particular we compute W_E as:

$$W_E(x_E) = \sum_{i=1}^{n} [l_i = x_E], \tag{2}$$

where l_i is the list of the unique edges, having the location as the prefix.

We compute W_L as:

$$W_L(x_E) = \sum_{i=1}^{n} [lS_i = x_E | lD_i = x_E], \tag{3}$$

where lS_i is the list of all source locations and lD_i is the list of all destination locations.

3.2 Dynamic Weight

The information related to the temporal evolution of the traffic for each location is especially useful in understanding which location to update (and therefore disconnect) first. The information is presented as a structured file that includes time series for each edge.

From an exploratory analysis of the aforementioned, the time series present an intra-day seasonality as well as a weekly seasonality (Figure 2). The objective of the algorithm used for this data is therefore to find the perfect time for each location that impacts the traffic the least.

In the specific, given a window of operation within the time series provided, for each location, we assign a weight based on how much provisioning would affect the location. In our work, we divide the weights as optimal, suboptimal, acceptable, and irrelevant (i.e. 3, 2, 1, 0). This means that once we find the minima we assign to that specific time frame the value 3. Following, we compute the minima again, but this time within the time series having the previously time-frame dropped; we assign the value 2 to the newly found minima(s). We repeat the same procedure and we assign the value 1 to those that are found as 3rd minima. We assign 0 to the remaining time frames.

3.3 Cluster ID

A fundamental part of the algorithm is related to the process of clustering the different locations. To compute the clustering we rely on the python package NetworkX [8], used for the analysis of complex networks. NetworkX is mostly known in the literature for its ability to create a visual representation of a network, however, one of its less known but powerful strengths is the ability to compute cluster coefficients. In short, the cluster ID is the ID assigned to a location and used for grouping the locations sharing the highest number of edge connections between them.

The cluster coefficients have therefore been computed through the NetworkX Clustering function giving as an input W_L and the maximum number of allowable clusters. Once the coefficients are computed we created the clusters by computing evenly spaced areas and assigning each area based on the coefficients computed in the previous step.

3.4 Smart Edge Scheduling Algorithm

Given two files related to Topology (TN) and to the temporal evolution of the traffic (TS) we develop our algorithm as shown in Algorithm 1.

More in detail, for each of the possible i locations (l_i) in TN, we compute both the Edge-Based weight (W_E) and the location-based weight (W_L) as previously described in Equation 2 and 3 respectively. Once both of those are computed, we retrieve the static weight s_w for each possible location.

Then, by making use of W_L, we compute the cluster numbers using NetworkX.

Following, for each specific location, we assigned the dynamic weights DW by finding the minima (first, second and third) in TS. This means that the time frame with the minimum amount of data sent will have the highest dynamic weight assigned (3), the second minimum will have the second highest dynamic weight assigned (2), and so on until the weights are assigned.

Once the dynamic weights are assigned, we sort s_w from the lowest to the highest value. The reason behind this choice is that we want to prioritize locations that have the less amount of edges and connections as we will have fewer chances to redirect the connections to adjacent edges and therefore, impact more users.

Then for each location x_E in s_w we search for the maxima in DW, and most importantly, the time-frame $(TF(x_E))$ when the maxima in DW is found. Once

Algorithm 1: Smart Edge Provisioning Algorithm

```
for $l_i$ in $TN$ do
    $W_E(x_E) = \sum_{i=1}^{n}[l_i = x_E]$;
    $W_L(x_E) = \sum_{i=1}^{n}[lS_i = x_E | lD_i = x_E]$;
    $s_w(x_E) = \alpha + W_E(x_E) + \beta * W_L(x_E)$;
end
$C = NetworkX(W_L)$
$DW = AssignDynamicWeights(TS)$
$SW = sort(sw)$                          ▷ From lowest to highest value of $x_E$
for $x_E$ in $SW$ do
    $TF(x_E) \leftarrow \max(DW(x_E))$
    if $SC(C(TF))$ is empty then
        $SC(C(TF)) = TF(x_E)$
    else
        $TF2(x_E) \leftarrow \max(DW(x_E), 2)$
        if $SC(C(TF2))$ is empty then
            $SC(C(TF2)) = TF2(x_E)$
        else
            $TF3(x_E) \leftarrow \max(DW(x_E), 3)$
            if $SC(C(TF3))$ is empty then
                $SC(C(TF)) = TF(x_E)$
            end
        end
    end
end
```

the $TF(x_E)$ has been detected we search if that specific TF has been assigned to any location within the same cluster C. If the TF for the specific cluster is vacant, then it is assigned to x_E, if not we repeat the same procedure for the second and third maxima. If all the possible detected TF have been already reserved, we move to the next location.

The reason for using TF is to maximize the degree of parallelism. We want to schedule inter-cluster parallel updates so that we have one update per node for each cluster, which means that the degree of parallelism depends the number of clusters created in the previous step.

Once all the locations have been served we have a clear schedule of which location should perform provisioning at each TF. On the other side, we will also have a list that reports which one is the correct TF to perform provisioning for each location. Inevitably, there are locations for which no suggested TF can be detected. This means that these locations (usually less than 5%), can be assigned to empty TF for their C without varying the impact.

4 Measuring the Scheduling Performance

To properly validate our algorithm it was fundamental for us to understand the performance of the model proposed. Moreover, it was important for us to take into account some key factors such as the number of intra-edge connections broken while provisioning, and the amount of data lost in the same phase. For this reason, we created two metrics based on such factors: the intra-edge impact and the traffic impact.

4.1 Intra-edge impact

The first factor to take into account when measuring the performance of the network is the number of multiple connections between different locations. A fundamental part of the algorithm relies on the creation of clusters composed of locations that are strictly related to each other. The reason behind the choice is to reduce the number of parallel unavailable locations which share multiple connections. We know that different countries are composed of a different number of locations, therefore, countries with a higher necessity of connection are less demanding. For this reason, we need a factor that shows the ability of the proposed algorithm to keep dense active connections alive when the throughput is high and heavily penalize situations where suggested scheduling cannot be proposed.

The intra-edge impact takes as input the proposed scheduling and the topology of the network: the first provides information related to *when* a specific location is shut down, while the second about *which* connections are going to be impacted by the provisioning. To penalize a situation where scheduling was not possible, all the locations without a suggested schedule are grouped in the same time frame.

4.2 Traffic impact

The second factor to take into account when measuring the performance of the network is the amount of data lost during the provisioning. The goal of the algorithm is to minimize such an amount through optimal scheduling so that precise handovers can be performed and the chance of failure is reduced to the minimum.

In our environment, the information related to the traffic is provided in time frames of 15 minutes each. The provisioning time is set so that out of 30 minutes required, for the first 10 minutes (i.e. 1/3 of the time) the system runs at lower capability and tries to perform handovers, during the following 5 minutes the system is inaccessible, and for the last 15 minutes the location runs again at lower capability.

Knowing this, we try to schedule handovers during the whole provisioning time, however, we know from the literature [9] that usually 20% of handovers fail. When creating the traffic impact factor, we grouped all of this information and created a factor that takes as input the proposed scheduling and the temporal evolution of the traffic. This means that for each specifically scheduled provisioning in the first input we search a correspondence for the TF, once the traffic information is found we store it as $v1$ and the following TF as $v2$. Then,

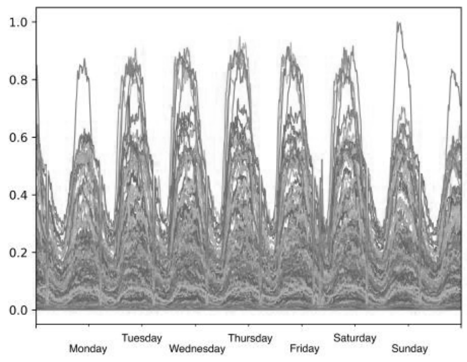


Fig. 3. Seasonality of the active traffic for all the locations considered.

we compute the traffic impact for the specific location as:

$$TI = \frac{\frac{1}{3} * v1 + (\frac{2}{3} * v1 + v2)/5}{5} \tag{4}$$

This means that we try to perform handovers all the time. However, statistically, 1 out of 5 times the handover fails and we need to compute the value we would lose in such an event. The traffic impact is composed of two parts, in the first we compute the complete outage which is taking one-third of the time of $v1$, in the second we compute the partial outage, which is taking two third of the time of $v1$, and the full $v2$. In our environment, during a partial outage the system can run at 80%, which means that during that period we lose 1/5 of the traffic.

5 Validation

To validate our scheduling algorithm, we used a system composed of 800 locations, with 270,000 edge nodes in total, averaging around 336 edge nodes for each location.

First of all, we calculated the static weights as described in 3.1. We obtained a value for each location; such value is only depending on the locations and the Edge devices, therefore, it is not changing, unless the architecture of the network itself would change.

Following, we calculated the dynamic weights described in 3.2. As it can be seen from Figure 3, there is a clear seasonality in the data, which allowed us and consider the temporal evolution of the system and therefore calculate the dynamic weights, for each of the locations.

Once the static and dynamic weights have been calculated, we calculated the cluster number using NetworkX as described in 3.3. This allowed us to have a modeled representation of the edge nodes and their location.

Our smart edge update scheduler produced therefore a proposed update scheduling. In Table 1 it is possible to see an example of the scheduling for some of the locations. As it can be seen, for each of the locations we have a time that represents the moment in which the update is scheduled.

Table 1. Update scheduling example.

Location	**Update Schedule**
235	2022-03-30 02:15:00 UTC
268	2022-03-30 02:30:00 UTC
224	2022-03-30 02:30:00 UTC
318	2022-03-30 02:45:00 UTC
362	2022-03-30 02:45:00 UTC
388	2022-03-30 02:45:00 UTC
402	2022-03-30 03:00:00 UTC
455	2022-03-30 03:00:00 UTC
469	2022-03-30 03:00:00 UTC

6 Conclusion

In this work, we provide a smart edge provisioning algorithm to minimize the number of dropped services and maximize the quality of service in a service-oriented architecture. We developed this algorithm to tweak at its best the environment provided resulting in a reduced amount of time necessary to perform a full upgrade of the elements composing the network and, therefore, not impacting availability of service in the hours with the highest demand. Such an environment is composed of a worldwide network divided into multiple locations, each one composed of multiple EDGE devices providing instruction to multiple IoT devices.

Ideally, the best possible outcome would be to update all the available locations without dropping any information provided to the IoT devices. This would mean that when performing the provisioning (update and restart), the amount of IoT devices connected to the location would be 0.

A possible way to achieve this condition would be by performing handovers to different locations. When performing handover we need to take into account two conditions:

– Network availability: when performing handover we need to make sure that the neighbor locations which are going to provide the service to the IoT devices will not be overpopulated by those. This would risk the malfunctioning of two locations.

- Failed handover: as described in Section 4.2, we know for sure that an average of 20% of handovers fail.

Therefore, it is very important to search for the time frame where each location has the minimum amount of throughput.

Given the latter as an input, we developed an algorithm tailored to this environment and impact values to validate it.

As the singular location-based provisioning would be impractical due to a large number of locations, our first goal has been to understand how to cluster different locations so that those could be updated in parallel. For this reason, we calculated weights based on the topology of the locations (and the edge nested in those) and the throughput of each location at each specific time frame.

By jointly analyzing the calculated weights we were able to understand which of these locations have more importance (some locations have fewer connections to other locations and therefore need to be carefully provisioned), and at what time there is less stream of information in the whole environment.

Our algorithm suggests schedules for most of the locations in the network. For some of them, it was not possible to provide an optimal schedule for two reasons:

- impact: the amount of information that they transmit is lower compared to other locations.
- overconnected: they have a high amount of connections to other locations and therefore the handover is easier

For these locations, the overall impact on the network between upgrading in different time frames can be neglected.

Acknowledgments

This work was partially supported by the Adoms grant from the Ulla Tuominen Foundation (Finland) and the MuFAno grant from the Academy of Finland (grant n. 349488).

References

1. M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
2. N. Kherraf, H. A. Alameddine, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, "Optimized provisioning of edge computing resources with heterogeneous workload in iot networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 459–474, 2019.
3. X. Cao, G. Tang, D. Guo, Y. Li, and W. Zhang, "Edge federation: Towards an integrated service provisioning model," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1116–1129, 2020.

4. X. Xu, H. Cao, Q. Geng, X. Liu, F. Dai, and C. Wang, "Dynamic resource provisioning for workflow scheduling under uncertainty in edge computing environment," *Concurrency and Computation: Practice and Experience*, p. e5674, 2020.
5. M. C. Ogbuachi, C. Gore, A. Reale, P. Suskovics, and B. Kovács, "Context-aware k8s scheduler for real time distributed 5g edge computing applications," in *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2019, pp. 1–6.
6. J. Guo, C. Li, Y. Chen, and Y. Luo, "On-demand resource provision based on load estimation and service expenditure in edge cloud environment," *Journal of network and computer applications*, vol. 151, p. 102506, 2020.
7. C. Li, J. Bai, Y. Ge, and Y. Luo, "Heterogeneity-aware elastic provisioning in cloud-assisted edge computing systems," *Future Generation Computer Systems*, vol. 112, pp. 1106–1121, 2020.
8. D. A. Schult, "Exploring network structure, dynamics, and function using networkx," in *In Proceedings of the 7th Python in Science Conference (SciPy*. Citeseer, 2008.
9. P. Legg, G. Hui, and J. Johansson, "A simulation study of lte intra-frequency handover performance," in *2010 IEEE 72nd Vehicular Technology Conference - Fall*, 2010, pp. 1–5.

PUBLICATION V

MLOps pipeline development: the OSSARA use case

Sergio Moreschini, David Hästbacka, and Davide Taibi

In: *Proceedings of the Conference on Research in Adaptive and Convergent Systems*. 2023. In press.

MLOps Pipeline Development: The OSSARA Use Case

Sergio Moreschini
Tampere University
Tampere, Finland
sergio.moreschini@tuni.fi

David Hästbacka
Tampere University
Tampere, Finland
david.hastbacka@tuni.fi

Davide Taibi
University of Oulu,
Tampere University
Oulu, Finland
Tampere, Finland
davide.taibi@oulu.fi

ABSTRACT

Software development based on MLOps practices is entering its early adoption stage. As for it, practitioners and researchers are starting to develop pipelines composed of tools capable of automating the whole software lifecycle. The development of the pipeline however is not as straightforward as it looks and some key points need to be addressed. In this work, we propose our vision for the development of the MLOps pipeline by showing what to keep into account when choosing the tools for each step of a pipeline. The proposition has been backed up by describing a developed use case scenario: the OSSARA use case. We believe that the presented use case, as well as the remarks presented for the process of tool selection for each MLOps phase, will help practitioners and researchers in the process of developing their own pipelines.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

MLOps, Machine Learning, pipelines

ACM Reference Format:
Sergio Moreschini, David Hästbacka, and Davide Taibi. 2023. MLOps Pipeline Development: The OSSARA Use Case. In *International Conference on Research in Adaptive and Convergent Systems (RACS '23), August 6–10, 2023, Gdansk, Poland.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3599957.3606211

1 INTRODUCTION

The increased adoption of Machine Learning (ML)-based software has inevitably created a need for a new method to efficiently develop software based on this new trend. Even if the term MLOps (i.e. Machine Learning Operations) has been widely accepted in literature, the concept does not provide uniform guidelines to develop ML-based software.

As a direct evolution of DevOps, it inherits some of its fundamental concepts such as Continuous Integration (CI), Continuous Delivery (CD), and the automation derived from such concepts. Some of these concepts take a step further compared to DevOps by adding, for example, new levels of automation such as Continuous Training of the underlying ML code behind the software.

The best way to achieve complete automation in MLOps is to create pipelines capable of automating the process of developing, testing, deploying, and monitoring ML models at scales. Such pipelines are created to meet the goals set by MLOps of ensuring the quick delivery of high-quality models as well as the cooperation among software and ML developers on one side, and IT operations on the other.

In order to optimize the process of MLOps pipeline development it is essential to have a full understanding of both the underlying ML code and the results that need to be achieved. To do so it is also essential to have the right set of tools to ensure full compatibility among the different MLOps phases covered.

In this work, we aim at providing some insights related to the process of choosing and combining tools for creating MLOps pipelines. In particular, we want to share our experience when developing an MLOps pipeline based on a model for OSS Abandonment Risk Assessment, namely OSSARA.

The work provides a different perspective on the development of a software-based MLOps pipeline. Contrary to most works that focus on a definition of MLOps based on the different levels of automation provided in [2], we focus our attention on reducing the gap between DevOps and MLOps by considering the latter as a natural evolution. A starting point is the selection of different open source tools based on the infinite MLOps pipeline as presented in [13]. The selection of such tools is also done taking into account the minimum number of usable OSS tools to create a meaningful pipeline for the tested use case.

Specifically, we aim to create a system that can perform continuous training on an ever-increasing dataset measured with different metrics and trained with different models, as we develop an MLOps pipeline for OSS abandonment risk assessment. The developed system will also need to be able to select the model with the best result and perform an automatic deployment of a new release on the basis of such a model.

The contribution of this paper is two-fold:

- **MLOps Pipeline Creation:** We provide insight into the process of developing an MLOps pipeline by focusing on the different stages of the infinite MLOps pipeline, which is derived from the original infinite DevOps pipeline.
- **Use case development:** We test the theorized pipeline by developing a specific use case based on OSSARA, the OSS Abandonment Risk Assessment model. Allowing the model to perform continuous training and deployment through REST API.

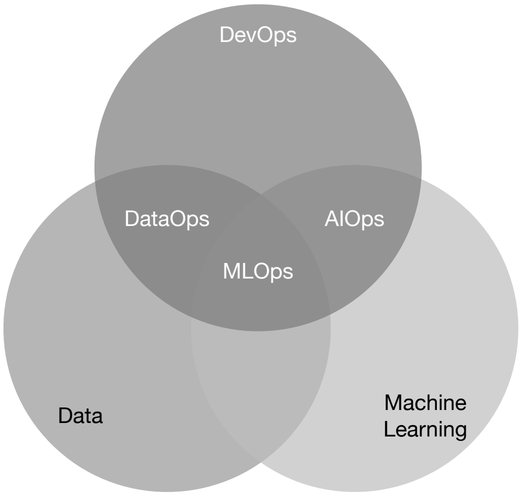


Figure 1: Different branches derived from DevOps

The results of this work will enable researchers and practitioners to gain insight into the process of developing fully automated ML-based software by making use of MLOps guidelines. In particular, we aim at increasing awareness of the importance of performing data versioning when developing ML-based software. On the other hand, the newly developed OSSARA model will achieve continuous updates through continuous training and deployment.

The rest of the paper is structured as follows. Section 2 covers the background starting from DevOps and reaching different branches such as MLOps. Section 3 presents the related works. Section 4 describes the process of selecting the most appropriate tool for each stage of the MLOps infinite loop. Section 5 introduces the original OSSARA model and describes how the risk assessment is calculated. Section 6 describes the process of creating a pipeline based on the OSSARA model. Section 7 provides some discussions while Section 7 concludes the work.

2 BACKGROUND

Since the dawn of software development, software engineers have been developing methodologies and guidelines to make the process of developing software a systematic process capable of ensuring specific levels of quality. The guidelines defined in the Software Development Lifecycle (SDLC) have been providing the correct methodology to follow for around 50 years.

With the evolution of the traditional software development model, the two main categories responsible for creating and maintaining the software, namely Developers and IT Operations necessitated a new set of practices aiming at breaking down the borders between such categories and boosting the collaboration among such entities. Such a set of practices has been defined as DevOps, a simple term coined based on the names of the two main leading actors responsible for such a revolution.

Even if there is no clear and uniform definition for DevOps has been set [9], all the definitions agree on 3 particular points:

(1) **What** - DevOps is a set of practices
(2) **Who** - combining software development and IT operations
(3) **Goal** - to deliver software quickly, rapidly, and efficiently.

Nowadays DevOps evolved by acquiring some of the main principles of other practices such as Agile development and CI/CD. From a different perspective, however, practitioners and technology experts focus more on the tools that allow providing the automation and the goals set by DevOps than the set of practices defined by DevOps itself. A clear example of such a shift is the periodic table of DevOps Tools a famous tool developed to identify the best tools for the software delivery lifecycle [4].

Different evolution of DevOps have been proposed and are receiving increasing attention. These new sets of practices are combining DevOps with the different entities on which is posed the main focus. A representation of DevOps and its branches is depicted in Figure 1. AIOps, for example, is defined as artificial intelligence for IT Operations and, even if places more attention on the Ops phase it is usually categorized as a branch of DevOps [6].

Among the different branches of practices derived from DevOps, we have the one that is focused on data: DataOps. In [12] Mainali et al. analyzed the different definitions of the term DataOps starting from what has been traced as the original definition in a blog post on the IBM Big Data & Analytics Hub titled "3 reasons why DataOps is essential for big data success[1]", which highlights the importance of executing data analytics rapidly. They also analyzed different works that focused on different perspectives of the topic dividing these into 3 main categories: goal-oriented, activities-oriented, and process and team-oriented. From the goal-oriented perspective, DataOps is mostly identified as a process (not a set of practices) to "eliminate errors and inefficiency in data management", from the activities perspective it is a set of practices in the data analytics field that combines DevOps and Agile methodologies. At last from the process and team-oriented perspective, it is defined as an approach for managing activities of the data lifecycle which requires collaboration among data creators and consumers.

A definition that seems to include all the previous definitions is the one presented in [5] by Ereth which defined DataOps as *"set of practices, processes, and technologies that combines an integrated and process-oriented perspective on data with automation and methods from agile software engineering to improve quality, speed, and collaboration and promote a culture of continuous improvement".*

All of the previous definitions do not take into account what has been made on the data, for this reason, it has been necessary to define another set of practices that are focused on data-driven applications, namely MLOps. Multiple entities have been trying to provide their vision on what are the most important stages and phases of MLOps, notably the most famous is the one from Google [2]. In the latter, the main focus has been posed on how to implement Continuous Integration (CI), Continuous Delivery (CD), and Continuous Training (CT) for ML systems. In particular, MLOps has been categorized based on the different levels of automation from 0 to 2, which respectively stand for Manual Process and CI/CD automation. An *"MLOps level 2: CI/CD pipeline automation"*

[1]Not available at the time of writing this work

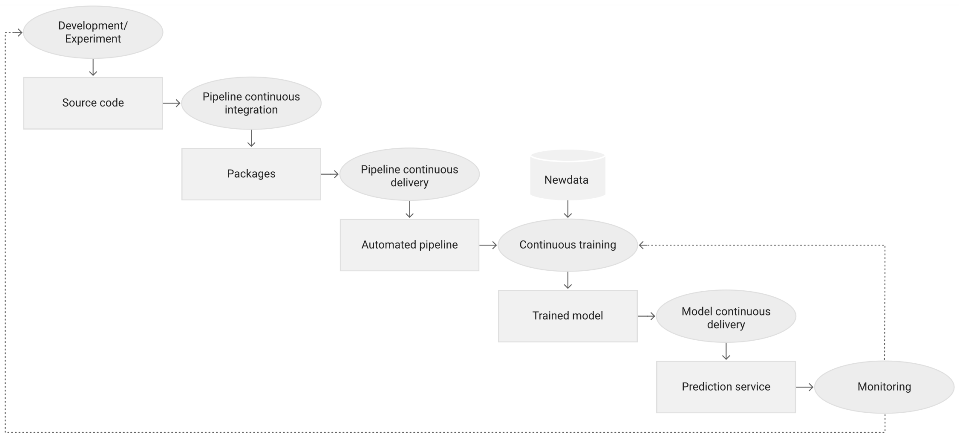


Figure 2: Stages of CI/CD automated pipeline. Reprinted with permission from [2]. (CC BY 4.0).

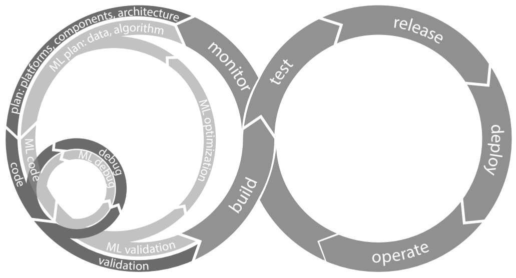


Figure 3: MLOps pipeline. Reprinted with permission from [13]. ©IEEE 2022.

is therefore capable of providing a system capable of *"deploying an ML pipeline that can automate the retraining and deployment of new models"*. Such a system is described by the pipeline depicted in Figure 2.

On the contrary, in [13] Moreschini et al. provided their vision of MLOps by highlighting the similarities and differences compared with DevOps. As such, the proposed pipeline is an evolution of the well know DevOps infinite loop, and as such it is composed of two main areas the Dev and the Ops (Figure 3). The main difference, in this case, is that the Dev part does not just include the work performed by the software developer but also the one from the ML developer which needs to work together to develop the software that will be passed to the IT Operations for the Ops phases.

Following, in the work [16], the authors provided a follow-up to their previous work by providing a list of tools capable of handling the various phases of the MLOps pipeline. Among the different tools provided, some of these have been categorized also as End-to-End Full-stack MLOps tools as they are capable of handling all the stages of the MLOps pipeline, while others have been categorized by the implementation that they can perform (i.e. CT, CI, CD, and OPS). Therefore, they suggest that it is possible to compose an MLOps pipeline by selecting one tool for each phase of the Dev phase and at least one tool for each phase of the Ops phase.

3 RELATED WORKS

In the last year, various works have tried to define a specific set of tools to define ML pipelines and provide their own examples related to how to use such pipelines in specific use cases. In [19], with the aim of presenting the potential of MLOps, the authors compared a specific set of tools for MLOps. After comparing them, in terms of preprocessing, training, management, deployment, and operations in ML projects, they presented an example of a specific workflow for "*Partially Automating Object Detection with MLOps Tools*". The paper bases its definition of MLOps on the different levels from [2], and the example aims to describe a level-1 automation using tools such as Git, GitHub actions, DVC, Hyperopt, and MLflow.

In [23], the authors combined the power of an end-to-end MLOps tool such as Kubeflow with additional tools such as Gitea and Drone to create an ML platform with a DevOps-capable framework. The framework was designed to continuously train several well-known deep learning neural networks, such as ResNet, MobileNet, and Inception V3, specifically to test the consumption in terms of energy and time.

In [18] an Edge MLOps framework has been presented. The framework uses a continuum-like system, as the CT phase is performed in the cloud through a cloud orchestration layer, while inference is performed on an edge device. The framework has been developed with Azure for ops-side management and Docker for deployment. The framework has been tested in a real-world scenario, where the goal was to deploy ML models to predict the air quality of rooms directly on the edge.

In [21], after reviewing the state of the art in MLOps and introducing MLOps as an evolution of DevOps principles, a use case was presented where modular pipelines were built for an ML time series forecasting system. Three approaches for the use cases were presented where an MLOps level 2 automated CI/CD pipeline is implemented [2]. Some of the tools that have been used are GIT, Jenkins, Docker, MLflow and Django.

4 DEVELOPING AN MLOPS PIPELINE

The process of creating an MLOps pipeline provides a very different approach compared to basic ML applications.

Plan: In this case, the selection of the tools used to perform the whole process is a vital step that needs to be taken into account already in the planning phase and, as part of the planning phase, it might variate along the development process.

Code/ML: Following the coding/ML phase in most cases is not usually associated with specific tools as the IDEs are only supporting the coding phase. Lately, there is an increased interest in different approaches to ML such as automated ML which is provided in some end-to-end MLOps tools such as Microsoft Azure Machine Learning, which automates the process of building machine learning models. For this reason, they can be defined as an extension of the library commonly used for creating ML architectures.

Build: For what concern the Build we find the biggest differences as we are not going to track the codes anymore but also its inputs and outputs. A huge part of creating an MLOps pipeline is to keep track of what are the different inputs associated with the various versions of the code and what are the outputs generated for such combination. In this work, this has been done by making use of the high compatibility between DagsHub [3] and DVC [8] for the input phase and between DagsHub and MLflow [17] for the output phase.

The importance of Data tracking is the fundamental part of DataOps and such similarity makes it often hard to delineate the difference between DataOps and MLOps. In [22], Tamburri indicates the *operation of Machine Learning continuously* as the borderline between DataOps and MLOps, specifically, MLOps needs to be composed of 5 specific functions:

(1) **Data ingestion/transport**: The layer at which the source data is collected on a regular basis and sent to the data warehouse or other such database, or ingested into the next layer.
(2) **Data transformation**: The layer that contains a set of transformations that support learning algorithms. It can include both preprocessing and data augmentation.
(3) **Continuous (re)Training** The layer responsible for training the network at regular intervals or when a certain event occurs.
(4) **Continuous Deployment** The layer responsible for deploying the system whenever the results achieved from the training phase overcome the previous training.
(5) **Output Presentation.** The layer responsible for constant monitoring of the deployed system and for presenting the desired results to the end-users.

This means that while both DataOps and MLOps are based on fundamentals of collaborative workflow, automation, and standardization, the main difference is that while it is possible to have DataOps without Machine Learning (and therefore MLOps) it is not possible to have the contrary as the second one relies on the first.

Test: The test phase is a fundamental step in the DevOps pipeline and is somehow a very ad hoc stage for what concerns MLOps as it is mostly dependent on the application that is tried to produce. First of all, when talking about ML, and mentioning test the main concept that comes to mind is to test how well the trained model performs on a dataset which has never been seen on the contrary when developing software the concept of testing is related mostly to ensuring the built software is defect free [10]. For its nature is therefore important to use tools to ensure the correct behavior of the system before encapsulating it and deploying it. From the algorithm development (ML) point of view tools such as DeepChecks [1] provide a continuous validation testbed to the ML system. On the other side, from the deployment (software) point of view, it might be necessary to run further tests due to the software nature of the system and the tools selected from the deployment stage. When choosing a tool for ML deployment it is quite normal to expect the integration of tools used to perform testing before deployment. However, extra testing can be performed by using libraries dedicated to it (for example PyTest in Python).

5 OSSARA

The OSS Abandonment Risk Assessment (OSSARA) has been first presented in [11]. The OSSARA model has been developed to provide a valuable tool to increase awareness of the problem of software abandonment. Such a problem is of particular interest in Open Source Software (OSS) components and libraries because such tools are developed in a collaborative, public manner.

Nowadays the process of composing software based on the adoption of *Components Off The Shelf* (COTS) is an effective method for software development. As OSS can be viewed as COTS, therefore, new software can be developed by creating custom code that aggregates different OSSs. In this particular environment, it is of vital importance that all the components of the software are reliable and stable, especially because the abandonment of OSS components could provide partial or full instability of the complete software. Among the main problems generated by the abandonment of an OSS there are security-related issues that might expose the users and the company creating the software.

The OSSARA model is calculated on 2 main factors:

(1) the likelihood of each component losing maintenance support,
(2) the importance of each component in the main system.

This means that starting from a system composed of several OSS components, it is analyzed the composition of the system to retrieve the weight of the component and, for each of these components, the abandonment probability is calculated in a specific time-frame (Figure 4). Formally this can be calculated as:

$$R_a = \sum_{m=1}^{k} w(O_m) * r(O_m) \tag{1}$$

where R_a is the overall abandonment risk of the system, $w(O_m)$ is the weight of the OSS component O_m and $r(O_m)$ is the risk that O_m will be abandoned. The method has been validated by evaluating

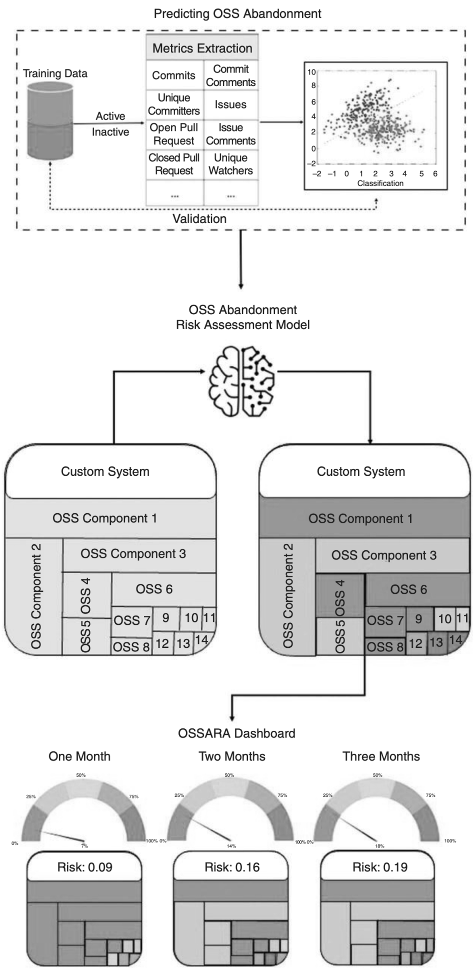


Figure 4: The OSSARA process. Reprinted with permission from [11]. (CC BY 4.0).

12,208 OSS projects that contain at least 1,000 commits from at least 5 unique contributors and are watched by at least 100 users. The classification has been performed using four different classification algorithms: decision tree, support vector machine (SVM), logistic regression, and naive Bayes. The results have shown that among these the logistic regression was providing the best results.

6 IMPLEMENTING THE OSSARA USE CASE

The OSSARA method has been used as a use case to develop an MLOps system. Following the list of tools suggested in [16], we created an MLOps pipeline to develop the OSSARA system in MLOps using Open Source Tools. In particular, such an MLOps system performs continuous training (CT) to compute the risk factor of each tool in the OSSARA system.

The implemented system will calculate the risk factors of each tool based on the model that is automatically chosen to be deployed. Such a model will be the one that provides the highest level of F1-score among all the models that have been trained with the different sets of the dataset and with the different classification algorithms.

The reason behind this choice is to simulate a system that is capable of performing continuous training to update its knowledge based on the current state of OSS projects of GitHub.

6.1 Continuous Training

One of the primary outcomes we tried to achieve in this development was the possibility of having a CT setup. To achieve such desideratum, we preprocessed the dataset used in OSSARA (https://doi.org/10.6084/m9.figshare.16944001. v1) so that it was divided into different time frames.

The different data frames have been created so that the starting data frame would include 2 years and half of the data for the first training, 3.5 for the second, and so on, generating in total 18 different sets for training.

The sets would then be loaded by the system sequentially so that it would simulate a continuously increasing dataset.

6.2 Tools Selection

The tools chosen to create this specific pipeline are DagsHub, DVC, and MLflow. For what concerns the developing part of the project DagsHub and DVC can provide all the desiderata. More into detail, even if both of these systems are categorized as *Build* tools, the cooperation of these tools can cover also different phases such as *Planning*.

To start the project, at first, a DagsHub repository has been initialized. Together with this, the DVC storage has been configured following the guidelines provided by the DagsHub repository. As previously specified the planning phase of the MLOps pipeline has been fulfilled by creating a *dvc.yaml* file which would create a data pipeline representation in the DagsHub repository. The generated representation is depicted in Figure 5.

Following the planning phase, we have the Code/ML phase. The code has been developed in Python, specifically using the scikit-learn library [20] in the Pycharm integrated development environment (IDE) [7] which has been connected to the DagsHub repository to simplify the stages of commit and push.

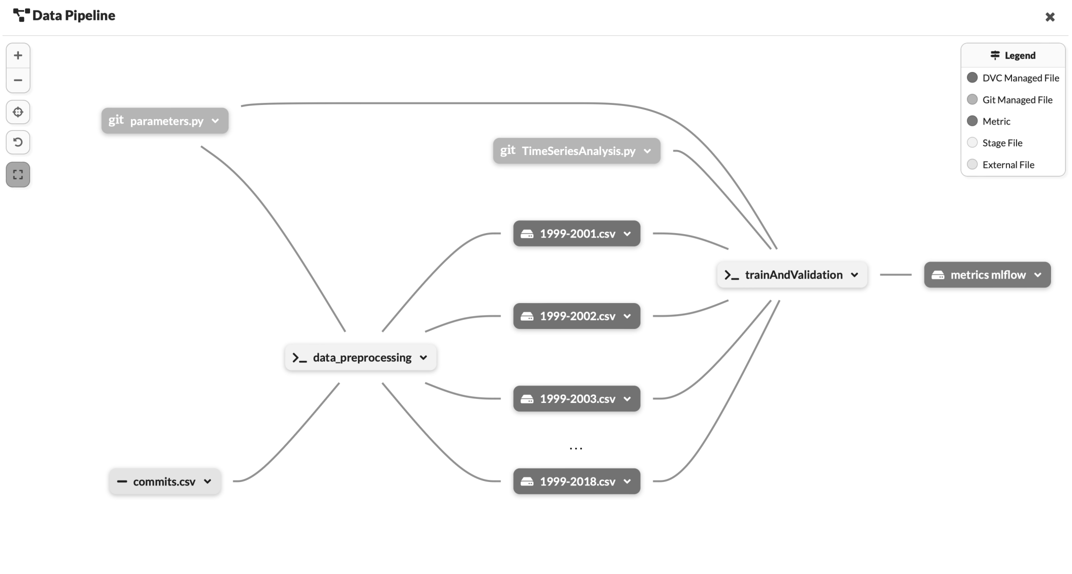


Figure 5: The data pipeline (Planning Phase).

As previously mentioned the phase of Build has been covered by both DagsHub and DVC. As the application does not require anything more than an API, the tools necessary for testing have been provided by the automatic testing performed by the deploying tool before performing the deployment. For what concerns data, as we are using a dataset already used previously we do not need to test it and perform drift analysis.

MLflow has been used to cover all the Operations phases. During the development of the code we integrated MLflow and we ran the experiments by following the MLflow guidelines in the DagsHub repository. Once the experiments have been produced it was possible to lunch the MLflow UI directly from the DagsHub repository, such UI provides information related to the different experiments, including all the metrics extracted (accuracy, macro, active, inactive) the duration, the source from which the experiment has been run and the model associated to the experiment.

A very interesting feature of MLflow is the possibility of deploying the model created as a REST API directly from MLflow, avoiding therefore to use of an external tool to perform such a step. This allows to invoke the local REST API endpoint with a POST input and after running the command it is possible to retrieve the inference results directly in the terminal.

6.3 Training and Deployment

Once the project has been developed, it was possible to perform a complete loop of the MLOps pipeline to measure the timing. A full training step of a single dataset was composed of 4 different algorithms. Each training step was based on the size of the dataset used to perform training. A summary of the timing for each training step has been reported in Table 1.

Once the training has been completed, it was possible to choose a specific run among those. Once the run has been detected, its id has been copied and used to perform a deployment based on REST API. The command used in the terminal was straightforward, as it was necessary to provide the MLflow URI, username, and password as shown in the experiment section of DagsHub together with the command MLFLOW MODELS SERVE –MODEL-URI RUNS:/*<RUN_ID>*/DT –NO-CONDA

The deployment has been immediate (3 seconds in total) and, as no public IP was provided the REST API has been deployed in the localhost port 5000 (as default). To invoke the REST API, a simple curl script with a POST input to the /invocations path was launched with an instant inference response.

6.4 Overall system architecture

The architectural design of the system implementing OSSARA is depicted in Figure 6. The system consists of 3 main components: the online repository, the server and a client.

The online repository, DagsHub, has been connected to DVC and MLFlow for data versioning and model registry.

The server is the one where the code resides, therefore at first the process of coding is completed and the practices of code versioning are finalized by pushing the different versions on DagsHub. Since the model resides online on DagsHub, it can be downloaded whenever it is needed. When the MLOps pipeline loop is started, if the data is not already on the server, it will be downloaded. During

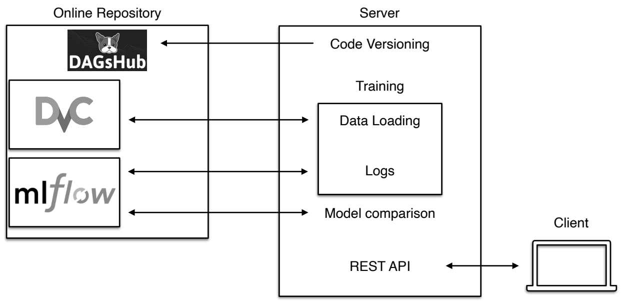


Figure 6: The overall system architecture

Dataset	Decision Tree	SVM	Logistic Regression	Naive Bayes	Total
1999-2001	5.1	4.6	5.6	4.5	**19.8**
1999-2002	4.9	4.7	4.5	5.6	**19.7**
1999-2003	4.9	4.8	4.7	4.5	**18.9**
1999-2004	4.8	5.0	4.6	4.9	**19.3**
1999-2005	4.7	4.8	4.6	4.9	**19.0**
1999-2006	4.7	5.4	4.7	4.7	**19.5**
1999-2007	4.7	4.6	5.3	4.8	**19.4**
1999-2008	4.9	4.8	4.8	4.7	**19.2**
1999-2009	4.9	4.9	4.7	4.8	**19.3**
1999-2010	4.8	4.9	4.7	4.5	**18.9**
1999-2011	4.9	5.2	4.9	4.7	**19.7**
1999-2012	4.9	5.7	4.9	4.8	**20.3**
1999-2013	4.9	6.0	4.7	4.8	**20.4**
1999-2014	5.0	7.2	4.9	4.8	**21.9**
1999-2015	5.1	8.6	6.0	4.4	**24.1**
1999-2016	4.5	10.0	4.8	4.9	**24.2**
1999-2017	5.2	10.0	4.8	4.8	**24.8**
1999-2018	5.1	10.3	4.7	4.6	**24.7**

Table 1: Timing for each training (in seconds).

the training steps, the different training logs are uploaded to the online MLFlow registry connected to DagsHub. Once the training is completed, the different models in the MLFlow registry are compared and once the best one is found, it is deployed through REST API.

Once the model is deployed, a client can now connect to the available port to submit the script and receive the inference from the server.

7 DISCUSSION

For the goal of this project, only part of the OSSARA use case has been developed as we were interested in simulating CT for the ML-based part of the model. The use case provided in the previous section provides an example of an MLOps pipeline based on the use of OSS tools. The use of such tools has been reduced to the minimum in order to provide a simple yet complete MLOps pipeline. It is important to specify that the current state of tools development is heavily based on the application that is intended to develop, and therefore smaller pipeline could be developed for different applications. An example is Computer Vision applications which nowadays have been heavily based on ML (and in particular deep learning) architecture and therefore have received ad-hoc tools such as Picselia or Roboflow.

A very important point that we would like to highlight by developing this work is the centrality of data in MLOps. This is because as the software to develop is heavily dependent on data it is fundamental to track any changes related to the input. For this reason, we see MLOps as a natural evolution of both DevOps and DataOps.

Thanks to this process it is now possible to automatically train and deploy a system capable of continuously monitoring the risk of embedding OSS components as COTS. We believe that such a process is of high interest for companies and developers who need to rely on OSS libraries and tools for developing their own software.

The system could be further implemented in the future by adding an alert system that could trigger messages for specific OSS tools or libraries inserted in a watch list to inform the users of the increased risk of abandonment for the element.

7.1 Threats to validity

We know that this work is subject to a variety of threats.

In particular, as stated in Section 7, we are aware that a different set of tools could be used to achieve the same results. The reason behind the choice of this specific set of tools is mainly based on the decision to create a free open-source pipeline that does not use any end-to-end tool but only relies on tools that are highly compatible and each of which partially covers the entire MLOps pipeline.

The results could have been influenced by the length of the different data frames used to simulate the CT setup. We know that in an ML-based system one, the result is heavily based on the dataset used to represent the information and therefore used to perform training. The reason behind the selection of such a specific length relies on the decision of having a time frame that is informative for a big part of the lifetime of projects.

8 CONCLUSION

In this work, we presented an approach to develop an MLOps pipeline. Specifically by making use of OSSARA model guidelines, we created a model that performs continuous training on the available dataset and automatically deploys the best model to perform OSS risk assessment analysis. The integration of OSS tools such as DugsHub, DVC, and MLflow allowed the development of a fully automated system capable of keeping track of the different data, models, and experiments.

The use case has been used to describe what is the process of creating an MLOps pipeline and discussing what are the fundamental steps to take into account when creating an MLOps system. In the specific the OSSARA model developed using the MLOps pipeline has gained some new characteristics from this fruitful development. Some of these are continuous updates through continuous training, automatic deployment based on the model with the highest F1 score, and the possibility to deploy the model independently of the device through REST API.

Future works include the implementation of new pipelines to address different problems, including, among others, time series analysis, fault detection, and computer vision. Moreover, it would be useful to extend this in the context of continuous [15] and cognitive [14] edge-to-cloud.

REFERENCES

[1] Deepchecks AI. 2023. Continuous ML Validation - DeepChecks. https://deepchecks.com. (2023).

[2] Google Cloud Architecture Center. 2023. MLOps: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning. (2023).

[3] Dagshub. 2023. Open Source Data Science Collaboration - DagsHub. https://dagshub.com. (2023).

[4] Digital.ai. 2023. Periodic Table of DevOps Tools. https://digital.ai/learn/devops-periodic-table/. (2023).

[5] Julian Ereth. 2018. DataOps-Towards a Definition. *LWDA* 2191 (2018), 104–112.

[6] IBM. 2023. What is AIOps? https://www.ibm.com/topics/aiops. (2023).

[7] IntelliJ IDEA. 2023. PyCharm: the Python IDE for Professional Developers by JetBrains. https://www.jetbrains.com/pycharm/. (2023).

[8] iterative.ai. 2023. Data Version Control - DVC. https://dvc.org. (2023).

[9] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. 2016. What is DevOps? A systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*. 1–11.

[10] Valentina Lenarduzzi, Francesco Lomio, Sergio Moreschini, Davide Taibi, and Damian Andrew Tamburri. 2021. Software Quality for AI: Where We Are Now?. In *Software Quality: Future Perspectives on Software Engineering Quality*, Dietmar Winkler, Stefan Biffl, Daniel Mendez, Manuel Wimmer, and Johannes Bergsmann (Eds.). Springer International Publishing, Cham, 43–53.

[11] X. Li, S. Moreschini, F. Pecorelli, and D. Taibi. 2022. OSSARA: Abandonment Risk Assessment for Embedded Open Source Components. *IEEE Software* 39, 04 (jul 2022), 48–53. https://doi.org/10.1109/MS.2022.3163011

[12] Kiran Mainali, Lisa Ehrlinger, Mihhail Matskin, and Johannes Himmelbauer. 2021. Discovering DataOps: a comprehensive review of definitions, use cases, and tools. In *DATA ANALYTICS 2021 The Tenth International Conference on Data Analytics*.

[13] Sergio Moreschini, Francesco Lomio, David Hästbacka, and Davide Taibi. 2022. MLOps for evolvable AI intensive software systems. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1293–1294. https://doi.org/10.1109/SANER53432.2022.00155

[14] Sergio Moreschini, Fabiano Pecorelli, Xiaozhou Li, Sonia Naz, Michele Albano, David Hästbacka, and Davide Taibi. 2023. Cognitive Cloud: The Definition. In *Distributed Computing and Artificial Intelligence, 19th International Conference*, Sigeru Omatu, Rashid Mehmood, Pawel Sitek, Serafino Cicerone, and Sara Rodríguez (Eds.). Springer International Publishing, Cham, 219–229.

[15] Sergio Moreschini, Fabiano Pecorelli, Xiaozhou Li, Sonia Naz, David Hästbacka, and Davide Taibi. 2022. Cloud Continuum: The Definition. *IEEE Access* 10 (2022), 131876–131886. https://doi.org/10.1109/ACCESS.2022.3229185

[16] Sergio Moreschini, Gilberto Recupito, Valentina Lenarduzzi, Fabio Palomba, David Hastbacka, and Davide Taibi. 2023. Toward End-to-End MLOps Tools Map: A Preliminary Study based on a Multivocal Literature Review. (2023). arXiv:cs.SE/2304.03254

[17] LF Projects. 2023. MLflow - A platform for the machine learning lifecycle. https://www.mlflow.org. (2023).

[18] Emmanuel Raj, David Buffoni, Magnus Westerlund, and Kimmo Ahola. 2021. Edge MLOps: An Automation Framework for AIoT Applications. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. 191–200. https://doi.org/10.1109/IC2E52221.2021.00034

[19] Philipp Ruf, Manav Madan, Christoph Reich, and Djaffar Ould-Abdeslam. 2021. Demystifying MLOps and Presenting a Recipe for the Selection of Open-Source Tools. *Applied Sciences* 11, 19 (2021). https://doi.org/10.3390/app11198861

[20] scikit learn. 2023. scikit-learn: machine learning in Python. https://scikit-learn.org/stable/. (2023).

[21] Rakshith Subramanya, Seppo Sierla, and Valeriy Vyatkin. 2022. From DevOps to MLOps: Overview and Application to Electricity Market Forecasting. *Applied Sciences* 12, 19 (2022). https://doi.org/10.3390/app12199851

[22] Damian A. Tamburri. 2020. Sustainable MLOps: Trends and Challenges. In *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. 17–23. https://doi.org/10.1109/SYNASC51798.2020.00015

[23] Yue Zhou, Yue Yu, and Bo Ding. 2020. Towards MLOps: A Case Study of ML Pipeline Platform. In *2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*. 494–500. https://doi.org/10.1109/ICAICE51518.2020.00102

S'io avessi, lettor, più lungo spazio da scrivere, i' pur cantere' in parte
lo dolce ber che mai non m'avrìa sazio;

ma perché piene son tutte le carte ordite a questa cantica seconda, non
mi lascia più ir lo fren de l'arte.

Io ritornai da la santissima onda rifatto sì come piante novelle
rinnovellate di novella fronda,

puro e disposto a salire alle stelle.

– Dante Alighieri, *Purgatorio*