Into the ML-Universe: An Improved Classification and Characterization of Machine-Learning Projects

Vincenzo De Martino, Gilberto Recupito, Giammaria Giordano, Filomena Ferrucci, Dario Di Nucci, Fabio Palomba

Software Engineering (SeSa) Lab — University of Salerno, Salerno (Italy)

Abstract

The prominence of Machine Learning (ML) systems led to the rise of Software Engineering for Artificial Intelligence (SE4AI), which addresses the unique engineering challenges of these systems. Researchers in SE4AI engage with three primary types of ML projects: those that apply ML techniques, those that develop new ML methodologies, and those that provide support tools and libraries. Current classification schemas distinguish ML projects based on their purpose and engineering quality, yet they miss a fine-grained classification of their nature and purpose. In this paper, we propose a novel, tool-supported automated classification schema for ML projects, coined MACHINE LEARNING AUTOMATED RULE-BASED CLASSIFICATION KIT (MARK), that builds on top of the work by Gonzalez et al. to refine the classification of applied ML projects into *'ML-Model Consumers,' 'ML-Model Producers,'* and *'ML-Model Producers & Consumers.'* We evaluated MARK through two empirical studies. The first assessed its classification accuracy across 4,603 ML projects from two datasets. The second analyzed repository metrics, such as community engagement, activity, and structure, to demonstrate MARK's potential in identifying trends and characteristics unique to each project type. Our findings indicate high F1-scores for our classifier, particularly for *'ML-Model Producer'* projects, though challenges remain for *'ML-Model Consumer'* classification. Significant differences in repository metrics among the classified projects highlight the usefulness of MARK, offering insights for researchers studying the socio-technical dynamics of ML projects.

Keywords: ML-enabled systems, Software Engineering for Artificial Intelligence, Empirical Software Engineering.

1. Introduction

Machine Learning (ML) has become a cornerstone of modern technology, playing a crucial role in a wide range of applications from healthcare to finance, and from autonomous driving to natural language processing [55, 61]. As the adoption of ML continues to surge, both practitioners and researchers are increasingly involved in the development, deployment, and analysis of ML systems [76, 60, 54, 50]. This rapidly growing field has catalyzed the emergence of Software Engineering for Artificial Intelligence (SE4AI) as a significant research area, focusing on the unique challenges and characteristics of engineering these complex systems [47].

In the realm of SE4AI, researchers primarily engage with three kinds of ML projects: (1) projects that apply ML techniques to solve specific problems, (2) projects that develop ML methodologies and models, and (3) projects that provide support tools and libraries to facilitate ML development. This tripartite categorization is crucial as it enables researchers to tailor their studies according to the specific nature and objectives of the projects under investigation.

Unfortunately, classifying projects according to this framework poses several challenges. First, the interpretation of project descriptions can be biased due to ambiguous or misleading information, which often lacks clarity or provides incorrect details [68, 75]. Second, the boundaries between categories can overlap, as projects may simultaneously apply ML techniques and develop methodologies, making it difficult to assign them to a single category. Third, the diversity in application contexts and the lack of a standardized taxonomy for ML projects contribute to inconsistencies

---

Email address: {vdemartino, grecupito, giagiordano, fferrucci, ddinucci, fpalomba}@unisa.it (Vincenzo De Martino, Gilberto Recupito, Giammaria Giordano, Filomena Ferrucci, Dario Di Nucci, Fabio Palomba)

and challenges in accurately categorizing these projects [72]. These challenges can contribute to interpretation bias, undermining the reliability of research findings and the development of best practices in the field. As a result, studies may become less comparable and risk overlooking critical aspects of the projects under investigation.

To exemplify the challenges and potential bias due to project misclassification, let us consider the study recently conducted by Calefato et al. [24], who aimed at assessing the support given by MLOps to the automation of ML projects. To perform this study, the authors required projects using ML techniques to address real-world problems so that they could assess the actual support provided by MLOps. As such, the authors identified 155 candidate ML projects relying on a pattern-matching strategy proposed by Biswas et al. [22]—this verifies the existence of keywords suggesting that the project exploits ML technologies. Upon conducting a follow-up manual inspection, Calefato et al. [24] found out that 105 of the 155 were misclassified as ML projects but did not contain ML components; 49 were ML projects defining libraries and/or producing new ML models, and only one was a genuine ML project useful for their investigation. According to their manual investigation, the misclassification was primarily due to buzzwords such as *"Artificial Intelligence"* or *"Machine Learning"* in the *README* file. On the one hand, the study shows that *misclassifying the nature of ML projects may lead to incorrect project sampling that significantly biases the interpretation of the conclusions drawn*. On the other hand, this example emphasizes *the need for approaches that may equip researchers with instruments to support the collection of ML projects based on their nature*.

Some attempts to mitigate those classification issues have been proposed so far. Widyasari et al. [73] released NICHE, *i.e.*, a dataset of ML projects classified based on their engineering quality, distinguishing between "well-engineered" and "non-well-engineered" systems. Gonzalez et al. [35] differentiated projects based on their purpose, such as whether they offer solutions to specific problems or provide toolkits for developing ML solutions.

While these classification schemas help provide an initial framework to understand the nature and purpose of ML projects, they would have only partially addressed the issues exemplified by Calefato et al. [24]. On the one hand, the classification by Widyasari et al. [73] would focus solely on the engineering properties of candidate ML projects. On the other hand, the classification by Gonzalez et al. [35] would mitigate the misclassification of non-ML projects but fail to differentiate ML projects that apply ML techniques to solve real-world problems or develop new ML technologies, thus lacking a *granular classification* of the ML projects. To illustrate the importance of granularity, consider two repositories commonly labeled as *'Applied AI and ML Projects'* under Gonzalez et al.'s classification schema. The first, Person-Detection-and-Tracking [1], demonstrates a complete pipeline for detecting and tracking individuals using computer vision techniques. This project involves training custom ML models, implementing object detection algorithms, and fine-tuning parameters, qualifying it as a *producer* of machine learning models, i.e., a project that uses ML libraries to create a custom ML solution. In contrast, the second repository, Onto2vec [2], applies pre-trained models to generate vector representations of biological ontologies. Rather than creating new models, it focuses on using existing ML models for knowledge representation and analysis, making it a *consumer* of existing ML solutions. Both repositories fundamentally differ in goals and technical requirements, yet current classification schemas might group them under the same broad category, *'Applied AI and ML Projects.'* This lack of differentiation may hinder their distinct purposes, complicating efforts to analyze project characteristics, compare methodologies, and draw actionable insights tailored to specific project types.

To deal with such a granularity issue, this paper builds on top of the classification proposed by Gonzalez et al. [35], proposing an automated heuristic-based mechanism, coined **M**achine learning **A**utomated **R**ule-based Classification **K**it (MARK), to classify GitHub Python projects at a finer-grained level based on Python APIs analysis: *'ML-Model Producer,' 'ML-Model Consumer,'* and *'ML-Model Producer & Consumer.'* Specifically, our mechanism is meant to be *combined* with the automated classification instrument proposed by Gonzalez et al. [35]: while these authors proposed a mechanism to distinguish between *'ML-Libraries & Toolkit'* and *'Applied Projects,'* our contribution focuses on refining the classification of the applied projects to provide a more detailed understanding of their nature, differentiating the way ML is used. This integrated classification procedure offers researchers a powerful tool to enhance the precision and relevance of analyses in SE4AI.

The potential value of MARK can be illustrated by reconsidering the challenges encountered by Calefato et al. [24]. Had MARK been available, it could have addressed the reported challenges by providing a more granular classification that distinguishes among *'ML-Model Producer,' 'ML-Model Consumer,'* and *'ML-Model Producer & Consumer'* projects. This finer granularity would have allowed the researchers to filter out irrelevant repositories more effectively and categorize the remaining projects according to their specific nature and purpose. Consequently, Calefato et al. [24] could have improved construct validity by ensuring that their dataset better reflected the intended

constructs of their study, i.e., real-world ML projects. Additionally, conclusion validity would have been strengthened, as their findings would have been based on a more accurately classified and representative dataset, thereby reducing biases introduced by manual corrections. Furthermore, MARK's automated approach could have significantly reduced the manual effort required to validate project classifications. Systematically distinguishing between different types of ML projects would have eliminated the need to rely on error-prone manual sampling methods, allowing researchers to focus on analyzing genuinely relevant projects. This improvement would save time and ensure greater consistency in project selection, ultimately leading to more reliable research outcomes.

We validate our classification schema through two empirical studies. First, we assess the accuracy of our classifier on 4,603 ML projects from two well-known datasets, NICHE [73] and the dataset provided by Gonzalez et al. [35]. Second, we characterize the classified Python projects using engagement and collaboration, activity, and structural metrics. While the first empirical study focuses on showcasing the capabilities of MARK, the second study aims to demonstrate the practical benefits of our classification schema for researchers. Specifically, if significant differences are observed between *'ML-Model Producer,'* *'ML-Model Consumer,'* and *'ML-Model Producer & Consumer'* projects in terms of the analyzed metrics, it would validate the importance of distinguishing and classifying projects based on their nature, thus highlighting the schema's value in enhancing the accuracy and relevance of research findings. The main findings of our work show that our classifier has high F1-scores, especially when classifying *'ML-Model Producers.'* As for the classification of *'ML-Model Consumers,'* we identify inherent challenges that may foster further research on the matter. Perhaps more interestingly, we identify significant differences among the classified projects in terms of the repository metrics considered, which demonstrate the usefulness of classifying Python ML projects according to their nature and purpose, other than providing initial insights into the socio-technical dynamics of ML projects. Particularly, projects belonging to the categories *'ML-Model Producer & Consumer'* and *'ML-Libraries & Toolkit'* exhibit notable differences in terms of various repository metrics, such as size, activity, and the number of contributors, compared to projects in the *'ML-Model Producer'* category. The findings suggest that *'ML-Model Producer & Consumer'* and *'ML-Libraries & Toolkit'* projects are often developed and maintained within larger and more complex environments, which may reflect their broader scope and greater complexity. We conclude our article by discussing the major implications of the study and the outlook for future research enabled by our work.

Structure of the paper. Section 2 discusses the related research and how current limitations motivate our work. Section 3 introduces MARK, our approach to classify machine learning projects. Section 4 describes the research questions driving our study. Section 5 presents MARK, detailing its design, validation, and limitations. Section 6 describes the research methods and presents the results of our second study, which characterizes ML projects using repository metrics. In Section 7, we discuss the major implications of our work. Finally, Section 8 wraps up the paper and discusses our future research directions in this area.

2. Related Work and Motivation

This section describes the previous related research papers, emphasizing how the current limitations motivate our work. For the sake of completeness, we organized the discussion to highlight how our work advances the current state of the art in the fields of SE4AI and automated classification of software projects.

2.1. Software Engineering for Artificial Intelligence

The work is motivated by and may contribute to addressing three main limitations in the current state of the art: (1) Understanding the nature of ML projects, (2) Clarifying the terminology associated with them, and (3) Mitigating the challenges of defining research studies focused on ML projects.

Previous research classified ML projects using different definitions and methods. In particular, Gonzalez et al. [35] classified over 5,000 GitHub repositories into "Applied AI and ML" and "AI and ML tools" categories: the former involves the use of Artificial Intelligence (AI) and ML techniques to solve specific, real-world problems, while the latter provides tools, libraries, and frameworks to develop, deploy, and manage AI and ML models. Although this high-level distinction offered an initial classification of ML projects, Rzig et al. [62] observed that over 30% of the projects in the Gonzalez et al.'s dataset [35] consisted of toy projects and study guides, indicating the need for more refined classifications. Compared to these previous papers, our work aims to provide a more detailed understanding

of the nature and purpose of ML projects, differentiating among projects that primarily apply ML techniques, those that contribute to advancing ML technology, and those that offer libraries and toolkits to implement ML techniques.

Widyasari et al. [73] recently published NICHE, *i.e.*, a dataset containing 572 ML projects. Using various metrics, the authors classified the projects as either "well-engineered" and "not well-engineered". While this classification represents a step toward understanding the engineering properties of ML projects, it does not provide insights into how the ML models are used. Martínez-Fernández et al. [47] provided a more inclusive definition of ML projects, termed "ML-enabled systems", which are projects that use at least one ML component. The ML model in these systems can vary significantly in its role: it may be relatively small and isolated, interacting with non-ML components, or it could be the core functionality of the system, with minimal supporting non-ML code. Additionally, an ML-enabled system typically includes components for training and monitoring models [52, 43, 39]. Compared to the work by Martínez-Fernández et al. [47], our work can be seen as complementary. We aim to refine and extend their classification by clarifying whether projects primarily apply existing ML techniques, contribute to developing new ML methodologies, or offer instruments to support the development of ML techniques.

Based on the argumentation above, our research outlines that existing work does not enable a granular analysis of ML projects. As illustrated in Section 1, this would notably impact the ability to differentiate between projects with fundamentally distinct purposes, such as those producing custom ML models, e.g., the PERSON-DETECTION-AND-TRACKING project [1], and those applying pre-trained models for domain-specific tasks, e.g., the ONTO2VEC project [2]. Without this granularity, important distinctions in project goals, engineering practices, or development characteristics are lost, hindering targeted analyses and actionable insights.

> ❶ **Limitation 1.** The state of the art lacks sufficient granularity, making it difficult to distinguish among projects that apply ML techniques, develop new ML methodologies, or provide supporting tools. This limitation hinders a detailed understanding of ML projects' objectives and contributions.

The terminology surrounding ML systems in real-world projects often lacks coherence. While terms like "ML-enabled systems," [47, 52, 71] "ML-infused systems," [41] "ML systems," [33, 44, 64] and "ML software," [35] provide a broad overview of such projects, their nature often overlooks the specific design and functionality of these systems. This inconsistency creates challenges in delineating the responsibilities, tools, and processes involved in producing, managing, and utilizing ML models. For example, Nahar et al. [52] highlighted this issue through interviews with professionals, showing how the lack of standard terminology complicates discussions around artifact responsibility, pipeline construction, model management, and other key areas of ML projects. As such, without clear and consistent definitions, researchers and practitioners may struggle to draw precise comparisons across studies or effectively communicate the roles and functionalities of ML systems. As a side contribution, our work has the potential to address this ambiguity by introducing a classification framework that distinguishes among projects that apply, develop, or support ML methodologies, thereby offering a clearer understanding of roles and contributions within ML projects, aiding in establishing a standardized terminology across the field.

> ❶ **Limitation 2.** The terminology for describing ML systems may hinder delineating roles and tools, impeding a precise understanding of the design and functionality of ML systems.

Research involving GITHUB entails significant effort due to the number of projects it stores [32, 40]. Researchers typically start by mining data using tools like the GITHUB API or GHTORRENT, employing specific search strings [30]. However, the initial volume of retrieved repositories often exceeds the scope of the study [40, 37, 35]; therefore, researchers must filter results using inclusion and exclusion criteria, removing irrelevant or problematic projects [40, 24, 62]. Errors in the filtering procedure may lead to biased outcomes or necessitate repeating data collection [37, 70]. Overly strict criteria can yield too little data, compromising generalizability or statistical significance [24, 21]. These challenges are emphasized when it comes to ML projects. For example, a recent study by Pepe et al. [57] analyzed PYTHON projects to investigate Self-Admitted Technical Debt (SATD) in deep learning systems. Consequently, their analysis targeted repositories implementing deep learning models, excluding projects that do not develop such models. Initially, the study collected a dataset of 160,365 repositories. However, after applying pre-processing steps to exclude tutorials, code books, toy projects, and repositories not written in PYTHON, the final dataset was narrowed to 100 projects. The time and effort required by Pepe et al. [57] to select a suitable analysis sample might have been significantly reduced by directly targeting projects based on a more granular classification framework. For instance,

a classification system distinguishing among *'ML-Model Consumers,' 'ML-Model Producers,'* and *'ML-Model Producer & Consumer'* could have allowed the researchers to focus exclusively on repositories classified as the latter two categories. This would have streamlined the sample selection process by filtering out irrelevant repositories, minimizing manual effort, and reducing the risk of biases. Indeed, depending on the scope of the analysis, researchers may want to focus solely on (1) projects that apply ML techniques, e.g., to analyze trends in ML adoption, predict software defects, (2) projects that develop ML techniques, e.g., to enhance model accuracy, investigate fairness, or (3) projects that support the implementation of ML techniques, e.g., to study the usefulness of libraries and toolkits. Our work addresses these challenges by providing a framework for classifying ML projects on GITHUB to offer researchers a more efficient way to filter and select relevant repositories for their studies.

> ❶ **Limitation 3.** Selecting ML projects from GITHUB based on predefined criteria can be time-consuming and error-prone, potentially leading to biased outcomes or insufficient data for analysis.

Based on these limitations, the goal of our work is refining the classification of ML projects, distinguishing among *'ML Libraries & Toolkits,' 'ML-Model Producers,' 'ML-Model Consumers,'* and *'ML-Model Producers & Consumers,'* whose union was defined as "ML Universe". More specifically, we define these categories as follows:

ML Libraries & Toolkits: Resources aiding developers in constructing or enhancing ML workflow. ML libraries are collections of pre-defined methods that allow developers to add ML functionalities to projects [45]. ML toolkits are external utilities that facilitate developers in developing ML pipelines. The category includes only libraries and tools that do not use or build ML models. An example project in this category is FASTAI [29], which allows developers to train deep learning models[1] and make inferences[2].Since our work seeks to refine the classification schema originally proposed by Gonzalez et al. [35], instead of creating an entirely new framework, we chose not to design new classification rules for this category. Instead, we deemed the original rules effective for accurately distinguishing this specific type of project.

ML-Model Producers: Projects building ML models *i.e.*, projects that invoke a training function. An example of an ML-Model Producer is the TUE ROBOTICS IMAGE_RECOGNITION [65] project, which contains packages designed for image recognition. The project allows training neural networks for recognition tasks *e.g.*, face detection, pose estimation, and color extraction using TENSORFLOW, OPENFACE, and OPENPOSE frameworks. In the work by Gonzalez et al. [35], these projects were categorized under the broad *'Applied Project'* category, making them indistinguishable from projects that merely utilize ML models. As such, our work aims at classifying them better.

ML-Model Consumers: Projects using ML models without building them. Such projects utilize external models through APIs to execute specific tasks, such as invoking GPT APIs for text generation or processing tasks. An example of ML-Model Consumer is the GPT2-DISCORD-BOT [10] project, which features a Discord bot leveraging GPT-2 model. Similarly to the previous case, Gonzalez et al. [35] categorized these projects under the *'Applied Project'* category. Our work aims at classifying them according to their specific nature.

By definition, these categories are *mutually exclusive* apart from an exception. A project can simultaneously be an *'ML-Model Producer'* and an *'ML-Model Consumer.'* This situation may occur when a project builds an ML model to carry out a certain function while using the built model to perform another task. In such cases, the project is categorized as an *'ML-Model Producers & Consumers.'*

2.2. Automated Classification of Software Projects

Broadening the scope of the discussion, our work contributes to the research on the automated classification of software projects. Previous researchers have primarily exploited machine learning solutions to classify projects according to their characteristics. For instance, Ugurel et al. [69] and McMillan et al. [49] exploited Support Vector Machine (SVM) and Latent Semantic Indexing (LSI), respectively, to categorize software projects according to their

---

[1]`https://docs.fast.ai/learner.html#learner.fit`
[2]`https://docs.fast.ai/learner.html#learner.predict`

scope, distinguishing among games, communication, compilers, etc. Similarly, Tian et al. [67] applied LSI to autonomously infer topics for software categorization. To train the machine learning algorithms exploited, these studies extracted features from software repositories, including descriptions mined from *README* files or information derived from the analysis of Application Programming Interfaces (APIs).

Two key observations emerge when comparing this research field with our work. First, like these previous studies, we rely on APIs for classification, focusing on how projects exploit external services. Our work complements existing research by providing further evidence that API-related information can assist project classification tasks. However, our work differs in its objective: rather than classifying projects based on their scope, we aim to classify them according to their usage of machine learning models. This introduces a novel application of API-related data, specifically tailored to understanding how ML projects interact with and implement machine learning models. For instance, we can characterize whether a project develops an ML pipeline rather than merely providing generic scope-based classifications, such as grouping repositories into categories such as games or communication tools. Contributing an approach that classifies API-related data becomes particularly relevant in ML contexts, where the scope of a repository is critical to understanding its design and functionality. Second, unlike previous approaches that used machine learning techniques, our method relies on heuristics. Our primary goal was to develop a classification tool that is not only accurate but also efficient, minimizing computational overhead and complexity. This heuristic-based approach allows for faster classification while maintaining high accuracy, particularly useful for large-scale analyses where speed is critical, e.g., mining software repository research. Therefore, our contribution is novel in its focus—classifying projects based on ML model usage—and its methodology, which prioritizes efficiency through heuristics rather than machine learning. Finally, it is worth remarking that API-based heuristics have also been successfully applied to a range of software engineering tasks, such as code quality assessment [25, 63], refactoring detection [46], and software maintenance prediction [17, 20]. While these studies focus on leveraging APIs to support specific tasks within a project, our work classifies entire projects based on their usage of APIs.

> ❶ **Limitation 4.** Existing API-based project classification methods do not specifically analyze how ML projects utilize machine learning models, nor are they optimized for large-scale analysis.
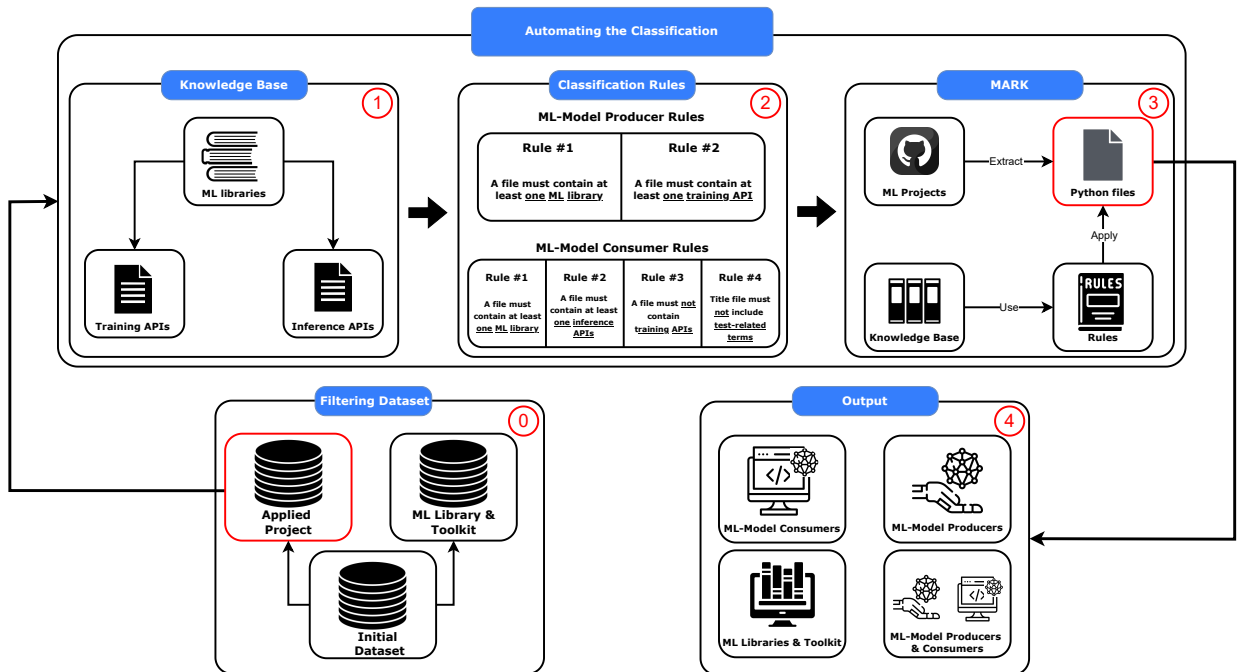

Figure 1: MARK: Overview of the Design.

3. MARK: Machine learning Automated Rule-based Classification Kit

In this section, we introduce MACHINE LEARNING AUTOMATED RULE-BASED CLASSIFICATION KIT (MARK), a novel approach for the classification of machine learning projects. MARK is based on heuristics, built from a Knowledge Base of APIs obtained on a filtered dataset. The following sections describe the design of the approach, while Figure 1 provides an overview of the major steps applied to design our approach.

3.1. Dataset Specification

Our work builds on the top of the study conducted by Gonzalez et al. [35], which introduced an automated classification procedure to distinguish between *'ML-Libraries & Toolkit'* and *'Applied Projects'* using natural language processing. We aim to further refine the classification of projects within the *'Applied Projects'* category. Consequently, the input for our approach is represented by a dataset that has been previously classified as *'Applied Projects.'* Currently, MARK focuses on categorizing applied projects containing components written in Python, given its success as a valuable and widely used programming language with many available ML libraries [59]. The focus on Python can be extended to include compatibility with other programming languages.
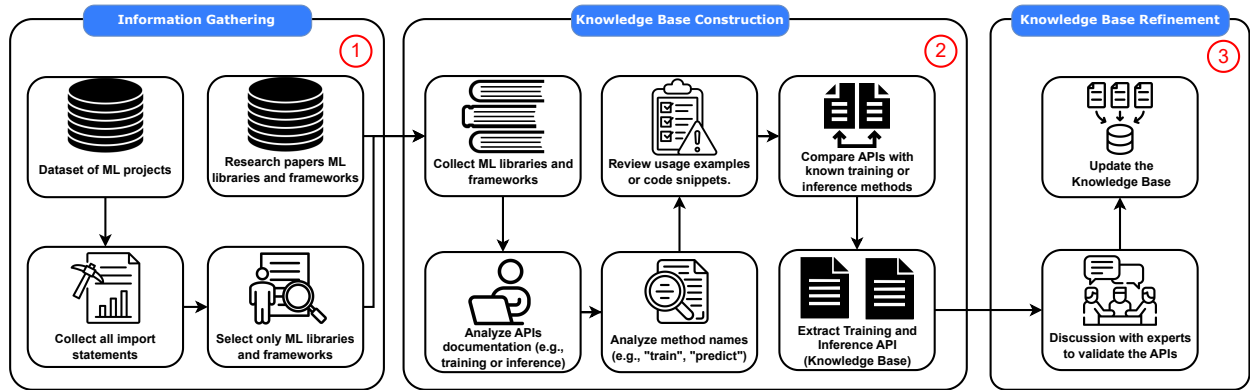

Figure 2: Overview of the Knowledge Base Construction and Refinement.

3.2. Definition of Knowledge Base

MARK leverages ML libraries and API definitions to classify ML projects. The Knowledge Base (KB) includes training and inference libraries and APIs. The definition of the KB involved three main steps, presented in Figure 2. As shown in the figure, the workflow began with conducting a preliminary *information gathering* phase (Step ① in Figure 2), whose aim was the identification of initial resources for extracting the KB. These initial resources included: (1) relevant research papers, i.e., in our case, the paper by Nguyen et al. [53] from which we derived the initial catalog of ML libraries and frameworks; and (2) state-of-the-art datasets of ML projects, i.e., in our case, those by Rzig et al. [62] and Widyasari et al. [73], which we used to collect additional ML libraries and frameworks. From a methodological standpoint, the first step was implemented through the following phases:

1. We collected relevant datasets and analyzed the literature to identify foundational or seed papers. We selected the one by Nguyen et al. [53] as the most suitable for our scope. However, should the KB need to be updated in the future, multiple papers, including ours, may be used as a starting point to snowball and find additional relevant resources, ensuring that all current catalogs of ML libraries and frameworks are considered. The initial catalog we considered [53] leveraged the most widely utilized PYTHON libraries and frameworks—specifically, the paper identified ten ML and Deep Learning (DL) libraries capable of training and making inferences with ML models. Nonetheless, when analyzing the work by Nguyen et al. [53], we realized that it could not comprehensively capture the current set of ML libraries and frameworks, since it was published in 2019. As such, we had to update this initial set, as described in the following phases. For future updates, however, the papers identified in the first step may already include up-to-date catalogs of ML libraries and frameworks. If this is

the case, a researcher interested in replicating our work may directly proceed to the second step of the KB workflow, i.e., *'Knowledge Base Construction.'*

2. To update the KB, we proceeded with a manual, tool-supported mining exercise against datasets of ML projects. Specifically, we developed a simple static analysis inspection tool to identify additional libraries and frameworks within our dataset. The tool was designed to scan each PYTHON file, extract all import statements, and organize them into a formatted spreadsheet, deleting duplicates and streamlining subsequent analysis and classification. Through this step, we collected all references to libraries `import` and `from` in 1,463 PYTHON files, finding 2,487 unique library imports. These import statements were then manually analyzed by the first author to determine whether they referenced libraries or served other purposes, e.g., importing local modules. Non-library references were excluded during this filtering process. Afterward, for each identified library, the first author manually compared it against the catalog provided by Nguyen et al. [53]. If the library was already available within the Nguyen et al.'s catalog, it was filtered out to avoid redundancy. Conversely, if the library was not present in their catalog, it was added to the set of libraries and frameworks considered in our study. As a final step, for each newly identified library, the first author manually collected its name and linked it to the corresponding official documentation or repository. This step enabled the cataloging of newly emerged libraries and frameworks, each linked to an established reference source. This ensured that the information was accurate, up-to-date, and reflective of the current state of the field, providing a reliable basis for subsequent analyses. The output of this first step was a refined and expanded list of ML libraries and frameworks, comprising 83 libraries, which served as the foundation for the second step of the workflow.

The second step, i.e., the *'Knowledge Base Construction'* (Step ② in Figure 2), began by examining the functionality of each library or framework to classify its APIs into categories such as training or inference:

1. The first author reviewed the official documentation for explicit descriptions of the API's functionality. For each library, he specifically searched for method APIs in its official documentation leveraging its website, *e.g.*, Gluon,[3] or its GitHub repository, *e.g.*, Anago.[4] The rationale for this step is that official documentation may indeed provide explicit descriptions of the API's intended functionality, e.g., whether an API is explicitly designed for tasks such as training models, performing inference, or supporting other ML-related operations. This step helped establish a baseline understanding of the core purpose of the API under consideration.

2. The first author analyzed method names for predefined keywords (e.g., *"train," "predict"*) aligned with common conventions in ML development. These keywords provided strong cues about the functionality of the API. For example, methods named `train_model()` are likely related to training tasks, while those named `predict_output()` may indicate inference operations. Identifying these patterns informed the subsequent classification of the APIs.

3. The first author reviewed code examples or snippets to verify operational behavior and functionality. For example, snippets that show the use of an API to fit models on training data or generate predictions can confirm its classification. This step was particularly useful when the documentation lacked explicit descriptions.

4. Finally, the first author compared the API with similar, well-documented training or inference APIs to assess functional similarity. For example, if an API demonstrates behavior analogous to TensorFlow's `fit()` method, it can be classified as a training API. This comparative approach added a layer of verification.

Based on the collected information, the first author categorized the 312 APIs from the 83 libraries as either *'training'* or *'inference'* APIs. This represented the initial version of the KB. Afterward, cross-checking with domain experts was essential to validate the results and resolve any ambiguities. This led to the final step, i.e., the *'Knowledge Base Refinement'* (Step ③ in Figure 2). These experts may include co-authors of the study, as in our case, or external collaborators with relevant domain knowledge. More specifically, the second and third authors of the paper jointly

---

[3] `https://cv.gluon.ai/contents.html`
[4] `https://github.com/Hironsan/anago/tree/master`

validated the categories assigned to each of the 312 APIs by the first author, using three predefined categories: "disagree," "further discussion," and "agree." If both authors agreed, the API was accepted without further discussion. In case of disagreement, a focus group meeting was held to decide whether to include the API as part of the KB. Following the validation, we computed Cohen's Kappa [27] to measure inter-rater agreement, which reached 0.90—indicating a high level of agreement among the three inspectors involved in the KB refinement process. As a result of the validation, 308 APIs were included in the KB, while four were excluded because they did not refer to training or inference mechanisms. The last step outputs a refined set of ML libraries and frameworks, each accompanied by classifying its APIs into categories such as training or inference.

3.3. Classification Rules

To build MARK we adopted a heuristic-based approach. The reason for this decision is twofold. First, heuristic methods provide greater transparency and interpretability in the classification process, making it easier to understand the rationale behind each decision. Second, given the nature of the problem and the relatively unexplored state of the art, we prioritized evaluating the performance of heuristic-based solutions due to their simplicity and efficiency. This was particularly important because our approach is designed to be scalable and suitable for large-scale analyses, such as mining software repositories. Had the heuristic-based methods shown inadequate performance, we would have considered more complex machine learning techniques, as recommended by prior research [34]. However, exploring these advanced techniques remains part of our future research agenda. As for the classification criteria, it is worth remarking that we were interested in defining classification rules that only discriminated among *'ML-Model Producers,' 'ML-Model Consumers,'* and *'ML-Model Producers & Consumers,'* as the input of MARK is represented by a dataset where *'ML Libraries & Toolkits'* projects are already classified. The classification rules were then defined according to the heuristics reported in the following:

ML-Model Producer: We examined API methods related to model training. A project was labeled as an *'ML-Model Producer'* if it contained an import statement for an ML producer library and an invocation of a related training method. Let F be a file belonging to the project P, let L be the set of ML libraries, and let A be the set of training APIs; we first defined the following two predicates:

$$\text{containsMLLibrary}(F, l) = \begin{cases} \text{True} & \text{if } F \text{ contains } l, \text{ with } l \in L \\ \text{False} & \text{otherwise} \end{cases}$$

$$\text{containsTrainingAPI}(F, a) = \begin{cases} \text{True} & \text{if } F \text{ contains } a, \text{ with } a \in A \\ \text{False} & \text{otherwise} \end{cases}$$

Starting from the definition of these predicates, we classify a project as *'ML-Model Producer'* if it meets **both** the following classification rules:

Rule #1: $\exists\, l \in L : containsMLLibrary(F, l)$ *i.e.*, a file must contain at least one ML library;

Rule #2: $\exists\, a \in A : containsTrainingAPI(F, a)$ *i.e.*, a file must contain at least one training API.

In other terms, if there exists at least one file F in the project such that F contains at least one ML library and at least one training API, the project was labeled as an *'ML-Model Producer'*.

ML-Model Consumer: We searched for API methods associated with model inference. A project was classified as an *'ML-Model Consumer'* if the library and the inference methods were identified. In formal terms, let the definitions of $containsMLLibrary(F, l)$, $containsTrainingAPI(F, a)$ be inherited from the *'ML-Model Producer'* classification context, let I be the set of inference APIs, and T be the set of testing-related words (*i.e.*, test, eval, example, and validate) we define the following predicates:

$$\text{containsInferenceAPI}(F, i) = \begin{cases} \text{True} & \text{if file } F \text{ contains } i \in I \\ \text{False} & \text{otherwise} \end{cases}$$

$$\text{containsTesting}(F) = \begin{cases} \text{True} & \text{if filename } F \text{ contains } \{\text{test, eval,} \\ & \text{example, validate}\} \\ \text{False} & \text{otherwise} \end{cases}$$

From the predicates, we classify a project as *'ML-Model Consumer'* if a file F in the project meets **all** the following classification rules:

Rule #1: $\exists\, l \in L : containsMLLibrary(F, l)$ *i.e.*, a file must contain at least one ML library;

Rule #2: $\exists\, i \in I : containsInferenceAPI(F, a)$ *i.e.*, a file must contain at least one inference API.

However, *'ML-Model Producers'* sometimes use the same methods as the *'ML-Model Consumer'* to test their models, which could increase the presence of false positives in our study due to possible misclassifications. To mitigate possible threats to validity, we applied two additional criteria:

Rule #3: $\nexists\, a \in A : containsTrainingAPI(F, a)$ *i.e.*, a file must not contain training API.

Rule #4: $containsTesting(F) = False$ *i.e.*, the filename does not include sub-strings related to testing.

In other words, if there exists at least one file *F* in the project such that *F* contains at least one ML library, at least one inference API, but the same file F does not contain a training API, and does not use the model only for testing, the project was labeled as an *'ML-Model Producer'*.

It is important to remark that the two sets of criteria that allow us to classify projects respectively in *'ML-Model Producer'* and *'ML-Model Consumer'* are not mutually exclusive. If a file *F* of a project *P* satisfies all the rules for classification as an *'ML-Model Producer,'* and another file *F'* of the same project *P* satisfies all the rules for classification as an *'ML-Model Consumer,'* then the project *P* is classified as an *'ML-Model Producer & Consumer.'* As an implication of the heuristics reported above, it is possible for MARK to classify an entire project as *'ML-Model Producer & Consumer,'* but not an individual file as *both* a producer and a consumer of ML models. This design decision is not due to the impossibility of a file theoretically serving as both a producer and a consumer. Rather, it is the result of a deliberate design decision in MARK to exclude this scenario. The heuristic rules were designed to assign each file a single dominant role (producer or consumer) to address a critical challenge in identifying real inference services within an ML project. Specifically, APIs that call a newly built model can serve two distinct purposes: actual inference (using the model in production) or evaluation during the training phase (testing its performance). Our key challenge was distinguishing between these two cases. To address this challenge, we *assumed* that a Python file maintains a consistent goal. If a file includes training APIs and invokes inference APIs, *we interpreted the latter as being used for evaluation rather than production-level inference*. This assumption was guided by the nature of ML practices observed during our preliminary analyses and the trade-offs involved in ensuring the precision of our approach. In the context of our empirical study, we analyzed the impact of the *file consistency assumption* on the classification performance of our approach (see Section 5).

As the reader may observe, the classification rules look pretty straightforward. While the rules themselves may seem simple, their design and implementation were anything but trivial. The true innovation lies in the systematic and effective application of API-related information to classify entire software projects. This process was far from linear; it required multiple iterations and refinements to ensure accuracy and robustness. As such, the technical novelty is not merely in the classification itself, but in how we harnessed API data in a comprehensive, scalable manner, capable of supporting large-scale analyses while maintaining a high level of precision.

3.4. Tool-Supported MARK Classification

To provide researchers with a readily-usable instrument, we implemented the classification rules into an automated instrument. This instrument, coined MARK, has two main functionalities: (i) cloning GitHub projects from a predefined list and storing their files locally, and (ii) automatically classifying the cloned ML projects. Specifically, the tool explores all Python files in a repository and applies the previously described rules to classify each project.

4. Research Questions and Methodical Overview

To overcome the limitations of the state of the art described in the previous section, the *goal* of the study is to refine the classification of ML projects, with the *purpose* of characterizing them better. The *perspective* is of researchers interested in understanding the nature of ML projects and having a framework to more precisely and efficiently identify study subjects based on the scope of their investigations.

Specifically, we have devised a two-stage investigation. In the first stage, we designed and evaluated a novel API-based, heuristic tool to classify ML projects named MACHINE LEARNING AUTOMATED RULE-BASED CLASSIFICATION KIT (MARK). As detailed in the following sections, MARK first collects existing datasets to acquire a knowledge base, which is leveraged to automatically classify ML projects according to their nature, distinguishing between projects that develop ML models *i.e.*, *'ML-Model Producers'* and projects that utilize ML techniques *i.e.*, *'ML-Model Consumer.'* This investigation addresses the following research question:

> 🔍 **RQ₁:** *How effectively can we classify ML projects according to their inherent nature and functionality?*

Once we had assessed the effectiveness of the automated classification mechanism, we performed a further step ahead to showcase the usefulness of the classification. In particular, we analyzed the differences between the classified projects using repository metrics, i.e., community engagement, activity, and contributor collaboration. Such an analysis does not aim to provide an exhaustive overview of the use of the classification but rather to provide insights into how to leverage it to understand project dynamics and community involvement. Should there be differences, we would have demonstrated the meaningfulness of classifying projects according to their nature and functionality, highlighting the potential benefits such classification may have for researchers interested in assessing the socio-technical dynamics of ML projects. As such, we addressed the following research question:

> 🔍 **RQ₂:** *How does the classification of machine learning projects relate to repository metrics?*

To report the results, we follow the guidelines by Wohlin et al. [74] and apply the *"General Standard"* recommended by the *ACM/SIGSOFT Empirical Standards.*[5]

5. RQ₁. Empirical Study: On the Performance of MARK

The *goal* of the first empirical study is to assess the performance of MARK in accurately classifying ML projects as either *'ML-Model Producer'* and *'ML-Model Consumer,'* with the *purpose* of validating the reliability and effectiveness of the classification schema. The *perspective* is of researchers and practitioners in the SE4AI field, who are interested in ensuring that the classification tool can be effectively applied in real-world scenarios, providing a robust foundation for further analysis and decision-making in ML project categorization.

5.1. Data Collection

Following our dataset specification defined in Section 3.1, we applied an extraction and filtering process to ensure compatibility to our approach. We started our data collection using two popular datasets. The first is a dataset proposed by Gonzalez et al. [35] and revised by Rzig et al. [62], which includes 4,031 ML projects. The second is NICHE [73], a dataset containing 572 ML projects. Combining them, we started our study with 4,603 ML projects. Before using these datasets, we applied a series of preprocessing steps to ensure consistency. Therefore, we integrated the two datasets and executed the following steps:

1. **Deletion of duplicate projects.** We first removed duplicated projects: specifically, we found 117 duplicates, which led to a set of 4,486 unique projects.

---

[5]`https://github.com/acmsigsoft/EmpiricalStandards.`

2. **Removal of Toy Projects and Tutorials.** We manually inspected the NICHE dataset to identify possible toy, empty, or tutorial projects. Please consider that we cleaned only the NICHE dataset as the dataset by Gonzalez et al. [35] has already been revised and cleaned by Rzig et al. [62]. More specifically, the first two authors analyzed the NICHE projects by reading the descriptions of their main pages on GɪᴛHᴜʙ and all websites linked to those pages. At the end of this step, 48 projects were removed.

3. **Collection of ML projects.** We cloned projects from GɪᴛHᴜʙ to verify their availability. We found that 92 projects had migrated to other version control systems or closed their repositories. By executing this step, we successfully collected 4,346 projects.

4. **Selection of ML-Applied projects**: This step aimed to distinguish all the considered projects between the two categories proposed by Gonzalez et al. [35], *i.e.*, *'ML-Libraries'* and *'Applied Projects.'* Since our approach focuses on the second category, we extended this categorization to the second dataset to ensure consistency across all analyzed projects. Specifically, we replicated the procedure described by Gonzalez et al. [35] and labeled each repository in the NICHE dataset [73] as either "applied ML" or "ML libraries." The first two authors manually classified the projects using the same methodology as Rzig et al. [62], originally applied to the Gonzalez et al. [35] dataset. Conflicts or unclassified projects were resolved through discussion, and to ensure reliability, the third author validated the classifications. Any disagreements were addressed in an additional meeting to finalize the categorization. As a result, 245 projects were classified as "ML libraries," and 157 as "applied ML."

Finally, we collected the set of applied ML projects. As a final result of the dataset extraction phase, we collected 3,005 *'ML applied projects'* and 1,349 *'ML-Libraries & Toolkits.'* To maximize the number of included projects, we did not impose additional constraints on factors such as years of activity, number of stars, or other characteristics, as these filters had already been applied [73, 35, 62].

Table 1: Statistical Metrics.

Metric	Formula
Accuracy	$(TP + TN)/(TP + TN + FP + FN)$
Precision	$(TP)/(TP + FP)$
Recall	$(TP)/(TP + FN)$
F1-Score	$(2 \times TP)/(2 \times TP + FP + FN)$

5.2. Data Analysis

To answer **RQ$_1$**, we aimed to understand how accurate MARK is in classifying projects through four well-known statistic metrics [58] (see Table 1). To validate the MARK, we created an oracle to manually classify each project into two categories: *'ML-Model Producer'* or *'ML-Model Consumer.'*

These categories were chosen because our approach specifically targets them, building on the work of Gonzalez et al. [35] and the *'ML Library & Toolkit'* has been manually classified, making further validation unnecessary.

To validate our approach, we considered the set of projects that we labeled as "applied ML" *i.e.*, we selected 3,005 projects. However, manually verifying the classification correctness requires high classification time to validate all of them manually. Therefore, we selected a representative subset for classification. In particular, we employed simple random sampling to produce unbiased samples from the dataset population [19, 66]. We utilized an online sample size calculator to draw statistically significant conclusions and estimate the necessary sample size.[6] This tool enabled us to determine the appropriate number of samples to maintain a margin of error of 5% and a confidence level of 95% and ensured that our sample size was sufficient to achieve reliable and valid results, thereby enhancing the robustness of our validation process. The first and second authors independently manually classified the oracle. For each project, they marked whether a project is an *'ML-Model Producer,'* an *'ML-Model Consumer,'* or both. The third

---

[6]https://www.qualtrics.com/blog/calculating-sample-size/

Table 2: Confusion Matrix for *'ML-Model Producer'*

		Predicted	
		Negative	Positive
Actual	Negative	91	1
	Positive	13	236

Table 3: Confusion Matrix for *'ML-Model Consumer'*

		Predicted	
		Negative	Positive
Actual	Negative	252	10
	Positive	25	54

author reviewed and verified the classifications for objectivity to ensure the oracle's validity and maintain unbiased judgment. We considered a statistically significant sample of 341 randomly extracted projects.

Besides the analysis of the MARK classification as a whole, we also conducted a step-by-step methodology to evaluate the contribution of each component of the approach. We began by assessing the impact of the knowledge base on a baseline version of MARK, which includes only Rules #1 and #2 for the classification of *'ML-Model Producer'* and *'ML-Model Consumer.'* This involved the comparison between the performance metrics obtained when using the original knowledge base from Nguyen et al. [53] with the enhanced knowledge base developed as part of our study. Afterward, we investigated the effect of the heuristic rules on the classification performance of *'ML-Model Consumer,'* focusing on the role played by Rules #3 and #4. In this respect, it is worth remarking that we could not investigate the effect of removing Rules #1 and #2 from the classification of *'ML-Model Producer'* and *'ML-Model Consumer'*: these rules are indeed the foundational heuristics for the classification of the projects. Rule #1 ensures that files contain at least one ML library, while Rule #2 enforces the presence of at least one relevant API (training APIs for producers and inference APIs for consumers). Without these rules, MARK would lack the fundamental criteria for distinguishing between producer and consumer roles, making it impossible to perform any meaningful classification. As such, Rules #1 and #2 are indispensable for the functioning of MARK and were locked in all configurations, serving as the baseline upon which additional rules and refinements were evaluated. As a consequence of these considerations, it is also worth remarking that we could not proceed with a finer-grained investigation of the heuristic rules composing the *'ML-Model Producer'* classification. This is because the classification relies solely on the two foundational rules and does not involve any additional heuristics. This explains why our analysis focused exclusively on the *'ML-Model Consumer'* classification, namely on the case where the impact of the additional heuristic rules could be meaningfully evaluated.

5.3. Analysis of the Results

In this section, we evaluate the performance of MARK in classifying ML projects. The evaluation is divided into three parts. First, we analyzed how MARK identifies the categories *'ML-Model Producer'* and *'ML-Model Consumer,'* discussing the misclassifications and their causes. This evaluation assessed MARK's capability to distinguish between these categories and identifies challenges and opportunities for improvement. Second, we conducted a configuration analysis to examine the contribution of each component and classification rule in MARK. This analysis helps determine which elements have the most significant impact on classification accuracy. Finally, we investigated whether MARK can recognize a new category (*i.e.*, *ML-Model Producer & Consumer'*) that combines the characteristics of *ML-Model Producer'* and *'ML-Model Consumer.'* The rationale for this approach is that classifying a project as both a producer and a consumer does not involve applying additional heuristics. Rather, it is a direct consequence of the existing rules used to classify projects as either producers or consumers. In other words, if a project contains files satisfying both sets of conditions rules, it is assigned to this dual-category. Consequently, misclassifications reported for the two base categories inherently propagate to errors in this combined classification. By adopting this reporting approach, we could first reveal the accuracy of the heuristics applied by our approach and then analyze how its misclassifications would impact the accuracy of the combined classification.

5.3.1. Assessment of 'ML-Model Producer' and 'ML-Model Consumer' Classification

Table 2 shows the confusion matrix for *'ML-Model Producers.'* The high number of true positives (236) and negatives (91) indicates MARK effectively identifies projects producing ML models. It returned only a false positive, demonstrating that the tool is unlikely to classify projects that do not produce any ML models as producers. However, the 13 false negatives show that the approach occasionally misclassifies producer projects as non-producers. This

Table 4: Statistical Metrics for *'ML-Model Producers'* and *'ML-Model Consumers'*.

Metric	*'ML-Model Producers'*	*'ML-Model Consumers'*
Accuracy	95.89 %	89.74 %
Precision	99.58 %	84.62 %
Recall	94.80 %	68.75 %
F1-Score	97.13 %	75.86 %

misclassification can occur due to projects using libraries that are not part of our APIs KB or building custom neural networks [9, 11]. Overall, the matrix highlights the strong performance of MARK in identifying ML Model Producers, which is critical for categorizing and analyzing these projects. Table 3 shows the confusion matrix for the *'ML-Model Consumer'* classification, highlighting a more challenging scenario. True negatives (252) and positives (54) indicate that the approach correctly identifies many non-consumer and consumer projects. However, 25 false negatives and 10 false positives highlight the difficulty of distinguishing *'ML-Model Consumers'* from other projects. The number of false positives suggests that the approach sometimes confuses projects with features similar to those of consumers. Additionally, the false negatives indicate cases where real consumer projects are not recognized. Several factors contribute to these misclassifications. Some libraries allow for predictions by directly entering the variable into the model without using specific inference APIs [6, 5]. The approach may miss projects using external libraries that are not included in the initial set [3, 8]. Projects sometimes create custom APIs for inference, which are not included in our KB. Our approach may incorrectly classify these as *'ML-Model Consumers'* since it detects the inference method but does not account for its actual usage context [13, 14, 4]. These issues highlight the need to refine our classification criteria to improve the accuracy and recall for identifying *'ML-Model Consumers.'*

Table 4 shows the statistical metrics for the classification performance of *'ML-Model Producer'* and *'ML-Model Consumer.'* For *'ML-Model Producer,'* the approach shows an accuracy of 95.89%, precision of 99.58%, recall of 94.80%, and an F1-Score of 97.13%. These high values indicate that the approach reliably identifies producer projects with low misclassification. In contrast, the metrics for *'ML-Model Consumer'* show lower performance, i.e., an accuracy of 89.74%, precision of 84.62%, recall of 68.75%, and the F1-Score of 75.86%. This result suggests that the approach struggles more with identifying *'ML-Model Consumer'* projects, particularly in terms of false negatives, which impact the overall recall of the approach. In this respect, a first observation to make is concerned to the relative importance that precision and recall have for the use case envisioned for MARK. We argue that the high precision of MARK is sufficient to meet its primary objectives. Since MARK is primarily intended for research, its high precision ensures that the projects it classifies are both reliable and accurate, which is crucial for the integrity of subsequent analyses. By minimizing false positives, MARK enables researchers to study ML project characteristics with greater confidence, reducing the risk of skewed results caused by misclassified instances. This precision is particularly valuable in research contexts where data quality and trustworthiness are paramount, ensuring that the findings can be confidently applied in practice. While the lower recall suggests that some projects may be missed, this is likely less concerning in a research context where a complete dataset is often unnecessary, i.e., our approach would still guarantee the selection of a significant and representative sample of projects for analysis. Although further refinements could enhance recall, we believe that MARK already effectively supports its research-oriented goals.

In any case, to better understand the causes of the classification errors made by MARK, the first three authors conducted a joint manual analysis of the misclassified instances. The goal of this collaborative effort was to evaluate the approach's errors, identify specific areas where the heuristic-based method may need refinement, and summarize the main causes of misclassification along with their relative frequency. Table 5 reports the causes of misclassification broken down by classification (*'ML-Model Producers'* and *'ML-Model Consumers'*) and further divided into false positives and false negatives. This division highlights how each cause specifically affects the respective classifications.

For *'ML-Model Producers,'* the *'Custom Solution'* category was the most common source of misclassification, with eight false negatives (2.35%). This suggests that projects implementing bespoke solutions for model training without relying on commonly recognized ML APIs pose a challenge to our classification approach. These custom implementations often lack standard markers, making them harder to detect using heuristic-based methods. Other

Table 5: Causes of Misclassification and their Frequency.

Reason Misclassification	ML-Model Producer		ML-Model Consumer	
	False Positive	**False Negative**	**False Positive**	**False Negative**
Custom Solution	0	8 (2.35%)	1 (0.29%)	15 (4.40%)
Keyword for API not exclusive for Inference	0	1 (0.29%)	3 (0.88%)	3 (0.88%)
Testing and Evaluation	0	0	7 (2.04%)	0
Do not use ML API	0	3 (0.88%)	0	2 (0.59%)
Library not included in our work	0	1 (0.29%)	0	2 (0.59%)
Both training and inference in the same file	0	0	0	2 (0.59%)
Training Keyword not used for training	0	0	0	1 (0.29%)
Not Machine Learning	1 (0.29%)	0	0	0
Total	1 (0.29%)	13 (3.81%)	10 (2.93%)	25 (7.3%)

sources of false negatives were *'Do not use ML API,'* with three false negatives (0.88%), and *'Library not included in our work'* with one false negative (0.29%). A single false positive (0.29%) was recorded in the *'Not Machine Learning'* category. Overall, the classification of *'ML-Model Producer'* projects led to a single false positive (0.29%) and 13 false negatives (3.81%), for a total of 14 (4.10%) misclassification cases.

For *'ML-Model Consumers,'* the most frequent misclassification was in the *'Custom Solution'* category, where projects implemented their methods instead of using the standard APIs, with 15 false negatives (4.40%) and one false positive (0.29%). *Testing and Evaluation* have five false positives (2.04%), where projects were misclassified as they used files solely for testing and evaluating models rather than consuming them. *Keyword for API not exclusive for Inference* have three false negatives (0.88%) and three false positives (0.88%) where keywords typically associated with inference were also used for other tasks, leading to misclassification, while *Do not use ML API* 2 false negatives (0.59%). Additional false negatives were found in *Both training and inference in the same file* (2 cases, 0.59%) *Library not included in our work* (1 case, 0.29%), and *Training Keyword not used for training* (1 case, 0.29%). Altogether, the classification of *'ML-Model Consumer'* projects led to 10 false positives (2.93%) and 25 false negatives (7.3%), for a total of 35 (10.23%) misclassification cases.

These results reveal that the majority of misclassifications were linked to the *'Custom Solution'* category for both *'ML-Model Consumers'* and *'ML-Model Consumers,'* with a higher number of cases observed in *'ML-Model Consumers'* (35) compared to *'ML-Model Consumers'* (14). The rare instances where MARK misclassified projects highlight specific edge cases that challenge the current heuristic-based approach. These findings highlight the opportunity for further refinements of our approach, including the integration of additional libraries and heuristics to effectively address these unique challenges and enhance the overall robustness of the classification process.

5.3.2. MARK Configurations

When it turns to the impact of each component on MARK, the results are reported in Table 6. The table presents the performance metrics—accuracy, precision, recall, and F1-score—under different configurations of MARK. In each section of the table, i.e., *"MARK 'ML-Model Producers' Configuration: Knowledge Base','* *"MARK 'ML-Model Consumers' Configuration: Knowledge Base,'* and *"MARK 'ML-Model Consumers' Configuration: Rules'* the first row represents the baseline configuration against which subsequent configurations are compared, allowing us to measure the incremental impact of each modification.

First, considering the impact of the knowledge base, we observed slight improvements in the performance metrics for both the classification of *'ML-Model Producers'* and *'ML-Model Consumers.'* Specifically, the enhanced knowledge base developed in this study, compared to the original knowledge base from Nguyen et al. [53], resulted in marginal increases across accuracy, precision, recall, and F1-score. For instance, in the case of *'ML-Model Consumers,'* recall improved by +1.20%, precision remained stable with a minor increase of +0.01%, and the F1-score rose by +0.64%. Similarly, for *'ML-Model Producers,'* we observed comparable trends, with slight yet consistent improvements in classification performance.

Table 6: Statistical Metrics for *'ML-Model Producers'* and *'ML-Model Consumers'* with different Knowledge Base and Rules (R) configurations of MARK. The symbol 🔒 indicates essential rules for classification.

Knowledge Base	R#1	R#2	R#3	R#4	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
MARK *'ML-Model Producers'* Configuration: Knowledge Base								
Nguyen et al.	🔒	🔒	-	-	95.01	99.57	93.60	96.49
Our dataset (MARK)	🔒	🔒	-	-	**95.89** (+0.88)	**99.58** (+0.01)	**94.80** (+1.20)	**97.13** (+0.64)
MARK *'ML-Model Consumers'* Configuration: Knowledge Base								
Nguyen et al.	🔒	🔒	✘	✘	74.78	47.89	85.00	61.26
Our dataset	🔒	🔒	✘	✘	75.66 (+0.88)	48.99 (+1.11)	91.25 (+6.25)	63.76 (+2.50)
MARK *'ML-Model Consumers'* Configuration: Rules								
Our dataset	🔒	🔒	✘	✘	75.66	48.99	91.25	63.76
Our dataset	🔒	🔒	✔	✘	86.80 (+11.14)	71.60 (+22.61)	72.50 (-18.75)	72.05 (+8.29)
Our dataset	🔒	🔒	✘	✔	76.54 (+0.88)	50.00 (+1.01)	88.75 (-2.50)	63.96 (+0.20)
Our dataset (MARK)	🔒	🔒	✔	✔	**89.74** (+14.08)	**84.62** (+35.63)	**68.75** (-22.50)	**75.86** (+12.10)

First, the results indicate that the enhanced knowledge base provided a more comprehensive and updated representation of relevant ML libraries and APIs, which reduced misclassifications. The improved recall demonstrates that the enhanced knowledge base enabled MARK to identify a slightly higher number of true positive classifications for producers and consumers. This improvement can be attributed to the inclusion of more recent and diverse ML components absent in the original knowledge base, thereby justifying the effort invested in creating the enhanced version. Although the scale of the observed improvements may be considered modest at first sight, these findings remain significant for two main reasons. First, the enhancements highlight the importance of periodically updating the knowledge base to capture the evolving landscape of ML tools and practices. Second, even incremental improvements in recall and F1-score enhance the overall robustness of the classification process, which is particularly valuable in scenarios where identifying as many relevant projects as possible is crucial. As such, these results highlight the practical value of the enhanced knowledge base in maintaining and extending MARK 's long-term utility. At the same time, the relatively minor scale of these improvements can be attributed to the fact that the foundational rules (Rules #1 and #2) already capture the most common and well-established ML libraries and APIs. As a result, the original knowledge base may have already addressed a significant portion of the classification needs. On the one hand, this highlights the value of the previous efforts by Nguyen et al. [53], as their knowledge base effectively captured essential elements of the ML ecosystem that are still relevant nowadays. On the other hand, this result provides insights into the long-term maintainability of the knowledge base. Our findings show that updating the knowledge base nearly five years after its initial inception led to only slight improvements in classification performance. This suggests that the most well-established ML libraries and APIs remain relevant over extended periods, with relatively few new additions being widely adopted. This observation implies that frequent updates to the knowledge base may not be necessary, as its foundational content continues to support effective classification. Consequently, our enhanced knowledge base emerges as a relatively long-lasting contribution that researchers can rely on without needing constant revisions.

In conclusion, these results provide additional value to the enhanced knowledge base, as they suggest it can serve as a stable and scalable resource for future studies. By requiring only periodic updates to incorporate major developments or new trends in ML libraries, our enhanced knowledge base offers researchers a reliable tool for analyzing ML projects while minimizing the maintenance overhead. Moreover, its longevity enhances its practical utility for a wide range of applications, further solidifying its relevance in the evolving field of ML project classification.

When it turns to the impact of the classification rules used to classify *'ML-Model Consumers,'* the results suggest that the inclusion of Rules #3 and #4 provide incremental stability to the classification capabilities of our approach. Specifically, the MARK version without Rule #3 achieves high recall (91.25%) but suffers from low precision (48.99%), indicating a significant number of false positives in the classification. The introduction of Rule #3 substantially improves precision to 71.60% (+22.61%) while reducing recall to 72.50% (-18.75%). These results suggest the validity of our assumption that files containing both training and inference APIs are more likely to leverage the latter for evaluation purposes rather than genuine inference. By excluding such files from the consumer classification,

Rule #3 effectively reduces misclassifications, particularly those resulting from overlapping producer-consumer characteristics. This trade-off between precision and recall reflects the heuristic's emphasis on minimizing false positives, which aligns with MARK's primary goal of ensuring the reliability and trustworthiness of its classifications. While recall decreases, the overall improvement in F1-score (+8.29%) demonstrates that Rule #3 achieves a better balance between the two metrics, reinforcing its role in enhancing the robustness of the classification process.

As for Rule #4, this is the rule that may have the most significant impact on false negatives, as its primary function is to exclude files that contain terms indicative of testing or evaluation (e.g., *"test," "eval,"* or *"validate"*) from being classified as consumers. While this exclusion reduces false positives by ensuring that testing-related files are not mistakenly identified as consumer files, it can inadvertently lead to false negatives in cases where such files genuinely serve inference purposes but happen to include these terms in their filenames. When analyzing the results shown in Table 6, we observe that Rule #4 slightly improves precision compared to the version of MARK that includes only Rules #1 and #2 (50%, +1.01%) but marginally decreases recall (88.75%, -2.50%). Despite these modest changes, the application of Rule #4 results in an improved F1-score (63.96%, +0.20%), indicating a slightly better balance between precision and recall. This suggests that Rule #4, while not drastically altering overall performance, plays a dual role: it reduces false positives by excluding files primarily dedicated to testing or evaluation but also slightly increases false negatives compared to the MARK version solely including Rules #1 and #2.

Finally, assessing the final version of MARK, which includes both Rules #3 and #4 alongside Rules #1 and #2 for the classification of *'ML-Model Consumers,'* we observed significant improvements in precision (84.62%, an increase of +35.63%) but also a notable reduction in recall (68.75%, a decrease of -22.50%) compared to the version of MARK that excludes Rules #3 and #4. This finding leads to two important conclusions that ultimately justify the design decisions to implement our approach. First, including all rules results in the best possible balance between precision and recall, as demonstrated by the highest F1-score (75.86%) achieved among all evaluated configurations. This suggests that the combined application of Rules #3 and #4 provides a robust framework for accurately distinguishing *'ML-Model Consumers'* by addressing false positives and false negatives holistically. Second, and more critically, the final version of MARK prioritizes maximizing precision, aligning with its intended design goal. As discussed earlier in this section, MARK is designed to support researchers in selecting a subset of ML projects for further analysis, particularly in scenarios where ensuring the correctness of the classification is paramount. High precision ensures that the identified consumer projects are highly reliable, minimizing the risk of including irrelevant or misclassified projects. By achieving a precision of 84.62%, the final version of MARK effectively minimizes misclassifications, which is crucial for studies requiring high-quality curated datasets. Although the notable decrease in recall indicates that some relevant consumer files are not identified, we deem this trade-off acceptable in the context of MARK's primary objectives. The ability to confidently classify projects as consumers outweighs the need to capture every relevant file, particularly when a representative sample is sufficient for most research applications.

5.3.3. Assessment of 'ML-Model Producer & Consumer' Classification

After collecting the results of the performance classification in individual projects, we evaluated the ability of MARK to assess its effectiveness in finding *'ML-Model Producer & Consumer'* projects. This category combines elements that characterize a project as a *'ML-Model Producer'* and a *'ML-Model Consumer.'* Table 7 shows the confusion matrix for detecting this category. Moreover, Table 8 summarizes the performance of our approach in detecting this category. Combining the information provided by the confusion matrix and the overall performance metrics, some considerations emerge. An accuracy of 90.62% demonstrates the strong overall reliability of the tool in identifying this category. The precision of 83.93% indicates that most positive classifications are correct, showing that the tool effectively identifies relevant projects while maintaining a low false positive rate. According to the confusion matrix, nine instances were misclassified as part of the *'ML-Model Producer & Consumer'* category. Additionally, the recall of 67.14% reflects the tool's ability to identify a significant portion of actual *'ML-Model Producer & Consumer'* projects, with 47 true positives and 23 false negatives. The recall score is strictly limited by applying the rules to detect *'ML-Model Consumer'* (*i.e.*, Rule #3 and Rule #4), as also denoted by similar scores. However, the F1-Score of 74.60% shows a balance between precision and recall, highlighting the tool's ability to identify *'ML-Model Producer & Consumer'* while offering opportunities for continued improvements. In summary, the performance of MARK in classifying *'ML-Model Producer & Consumer'* is closely related to the tool's ability to accurately detect individual categories of *'ML-Model Producer'* and *'ML-Model Consumer.'* In fact, the results for the new category

Table 7: Confusion Matrix for *'ML-Model Producer & Consumer'*

Actual		Predicted	
		Negative	Positive
	Negative	262	9
	Positive	23	47

Table 8: Performance Metrics for *'ML-Model Producer & Consumer'*

Metric	ML-Model Producers & Consumers
Accuracy	90.62 %
Precision	83.93 %
Recall	67.14 %
F1-Score	74.60 %

are well-aligned with the detection performance of the individual categories, reflecting the underlying consistency and effectiveness of the classification approach across individual and combined project categories. Therefore, the room for improvement observed in the detection of single categories is directly related to the misclassification of these projects in the *'ML-Model Producer & Consumer'* category. Enhancing the precision and recall for the *'ML-Model Producer'* and *'ML-Model Consumer'* projects will naturally lead to improvements in the classification of the combined *'ML-Model Producer & Consumer'* category.

> ↪ **Answer to RQ$_1$.** MARK could identify *'ML-Model Producer'* with a precision of 99.89% and a recall 94.80%. However, the performance is slightly lower for the *'ML-Model Consumer'* classification, which shows a precision of 84.62% and a recall of 68.75 %. Moreover, our approach can also classify the *'ML-Model Producer & Consumer'* category with 83.93% of precision and 67.14% of recall. Challenges in the classification process arise from custom implementations, non-standard APIs, and testing-related terms, which may contribute to misclassifications. Despite these limitations, the high precision aligns with MARK's primary goal of providing reliable classifications for research purposes. The detailed analysis highlights opportunities for refinement, including the expansion of the knowledge base and improved heuristics to address edge cases. Overall, MARK effectively supports its intended research-oriented use case, ensuring accurate and representative datasets for further analysis.

5.4. Threats to Validity

This subsection describes the main threats that can affect our first empirical study.

Construct Validity. We recognize that some libraries and methods may have been overlooked. However, our KB is based on the current state-of-the-art, specifically identifying manual inspection strategies for analyzing documentation and discovering inference and training methods. To mitigate this possible threat, we adopted a systematic approach to catalog libraries and methods and involved two or more inspectors to ensure the correctness and completeness of the identified APIs and methods. We conducted a review process between the first three authors for all decisions, with agreements and meeting sessions to ensure their inclusion. A total of 30 hours per author was required to perform these activities. To confirm the validity of these methods and ensure the competence of the classification activities, we describe the authors' experience. The first two authors are Ph.D. students with a major in ML and SE4AI and four years of experience in ML model development, while the third author is a postdoc with six years of experience in ML and SE4AI. The process is fully documented in the online appendix ("Pipeline" file).

Of course, we are aware that the KB may evolve with introducing new APIs and methods, which could potentially impact the applicability of MARK. Three considerations are relevant in this regard. First, this is a challenge inherent to all API-based approaches [49, 67, 69]. Second, while an outdated KB might reduce recall by missing newer APIs or methods, it does not directly compromise precision. Consequently, we are confident that even with some degree of obsolescence, our approach would continue to deliver reliable classifications, making MARK a dependable tool for research applications where precision is crucial. Third, our fine-grained analysis in Section 5.3.1 highlighted that the enhanced KB provided marginal but consistent improvements in precision and recall. These findings demonstrate the importance of periodic updates to maintain the relevance of the tool while underscoring that the foundational content of the KB effectively captures the core characteristics of ML projects. Thus, while regular updates are advisable, frequent revisions are not strictly necessary, ensuring MARK remains a long-lasting and practical resource for researchers.

Furthermore, our approach to classifying *'ML-Model Producer,'* *'ML-Model Consumer,'* and *'ML-Model Producer & Consumer'* was defined following a heuristic-based approach. This decision was driven by our design goals of achieving simplicity, efficiency, and transparency in classification, particularly given the need to handle large-scale data, as is often required in mining software repositories. Nonetheless, our results revealed that heuristics may occasionally fail to identify edge cases, such as projects implementing highly customized APIs or combining training and inference in non-standard ways. This limitation highlights the inherent trade-offs of a heuristic-based approach: increasing precision by focus on well-defined patterns often comes at the cost of reduced recall, as valid instances that deviate from these patterns are excluded. In this regard, we conducted additional analyses on the causes of misclassifications, along with investigating the impact of each heuristic on the overall performance of the proposed approach. These analyses revealed that the heuristics performed robustly for standard cases but struggled with cases that deviated from common patterns. These findings suggest that MARK could be further improved by integrating advanced natural language processing models capable of understanding the semantic context of API usage. This integration could help identify semantic patterns and reduce the likelihood of edge-case misclassifications. Additionally, incorporating a feedback loop to iteratively refine the KB and rules based on observed misclassifications could improve the robustness and adaptability of the tool over time. Such hybrid approaches, which combine heuristics with AI-based techniques, offer a promising direction to address the limitations observed in our current approach.

The last threat to construct validity concerns the implementation of our tool and its adherence to the identified detection rules. To ensure the tool's correctness, we defined test cases and conducted code review activities. Additionally, we assessed MARK performance by studying the contribution of each component to the resulting performance. Our evaluation of various configurations showed that while modifications to certain rules and the knowledge base enhanced precision, they occasionally diminished recall by omitting valid instances. This trade-off highlights the difficulty of balancing generalizability and specificity, emphasizing the need for iterative refinement of the rules and knowledge base to enhance MARK reliability across diverse design scenarios. For verifiability, we released the source code and all the material necessary to build it.

Conclusion Validity. In the context of $\textbf{RQ}_1$, we assessed the performance of the classifier based on the sample size. However, if, on the one hand, we selected a statistically significant sampling to conduct the empirical experiment, on the other hand, we cannot guarantee that the performances observed in the sample can be generalized to the entire ML project population. To calculate the performance of our tool, we built an oracle by manually labeling the statistically significant sampling in *'ML-Model Producer'* and *'ML-Model Consumer.'* Finally, to assess the tool's capabilities, we exploited well-established performance metrics [58]. Furthermore, we manually go deeper into the misclassifications of projects by manually analyzing projects and including the missed APIs and related methods. In addition, we completed our quantitative analysis with qualitative insights.

External Validity. We identified two main external validations. The first concern is the programming language used to investigate ML projects *i.e.*, PYTHON. We are conscious that other programming languages can be used in ML contexts; however, we focused on PYTHON due to its widespread adoption by practitioners worldwide. Additionally, many of the datasets released in the literature in ML contexts refer PYTHON projects. The second is related to performance generalizability. In this case, this threat is related to the quality of the dataset used. To mitigate this possible threat, we selected two well-known ML datasets *i.e.*, NICHE [73] and the Gonzalez's dataset et al. [35] and applied preprocessing strategies to remove possible toy projects, tutorials, and so on.

6. RQ$_2$. Empirical Study: On the Usefulness of MARK

The *goal* of the study was to understand the socio-technical characteristics of ML projects classified according to our schema, with the *purpose* of gaining insights into how the differences among these projects validate the usefulness of our classification. The *perspective* is of researchers who are interested in exploring the distinct patterns, challenges, and opportunities that emerge from different types of ML projects, ultimately aiming to refine best practices and improve the effectiveness of tools and methodologies in the SE4AI domain.

Table 9: Distribution of Projects by Category.

Category	Description	# Projects
ML Libraries & Toolkits	Projects that provide tools to develop ML solutions.	1,349
ML-Model Producers	Projects that invoke a training function.	1,359
ML-Model Consumers	Projects using ML models without building them.	56
ML-Model Producers & Consumers	Projects that simultaneously produce and utilize ML models within the same context.	493

6.1. Data Collection

The first research question evaluated MARK's ability to distinguish between *'ML-Model Producers'* and *'ML-Model Consumers'*. This evaluation was essential to validate the core functionality of MARK in correctly categorizing projects using machine learning models. However, the second empirical study expanded the scope to include all four project classes: *'ML-Model Producers,' 'ML-Model Consumers,' 'ML-Model Producers & Consumers'* and *'ML Libraries & Toolkits'*. This broader analysis enabled us to evaluate the overall effectiveness of integrating MARK with the original classification by Gonzalez et al. [35] as a research tool. It also allowed us to explore and compare the significant differences across all project categories, further demonstrating the utility of this combined classification approach. To achieve this goal, we essentially simulated a researcher's activities when conducting an empirical study comparing the characteristics of different ML projects. We began by using the datasets employed in RQ_1, specifically (1) the dataset initially proposed by Gonzalez et al. [35] and later refined by Rzig et al. [62], and (2) NICHE [73]. From these datasets, we applied MARK to classify projects labeled as *'Applied ML'* while retaining those labeled as *'ML Libraries & Toolkits'* as a separate category. Using MARK, we further identified projects related to *'ML-Model Producers'* and *'ML-Model Consumers'*. Subsequently, we added a step to identify projects classified in both *'ML-Model Producers'* and *'ML-Model Consumers,'* categorizing them as *'ML-Model Producers & Consumers'*. The final set of projects classified by MARK is in Table 9, which shows the distribution of projects among the four categories.

Afterwards, we computed repository metrics, following previous works [16, 51, 31]. These metrics described projects along multiple dimensions, characterizing the *socio-technical* dynamics arising in those communities. We collected data from these categorized ML projects using the GitHub API [12]. Table 10 details the metrics collected.

Firstly, we focused on **engagement and collaboration** metrics, including the *number of stars* and *number of forks*, which highlight user interaction levels. We also collected the *number of contributors* to gain insights into the collaborative nature of the projects and the extent of community engagement. Moreover, our data collection covered **activity metrics**, essential for evaluating each repository's activity rate and development efforts. Specifically, we gathered data on the *number of commits* and the *activity rate*, measured by the frequency of commits over time. Lastly, we included **structural metrics** to assess the overall complexity and maturity of the projects. We measured each repository's *size* regarding lines of code (LOC), which helps gauge the project's scale and potential maintenance demands. Additionally, we considered the *project age*, providing context on the project's maturity.

As already discussed in Section 4, it is worth remarking that this analysis is not intended to offer an exhaustive overview of the classification's applications but rather to provide insights into how it can be leveraged by researchers. The focus on project dynamics and community engagement stems from the availability of well-established metrics and tools for their measurement. Identifying any differences would validate the value of classifying projects based on their nature and functionality, thereby underscoring the potential benefits for researchers interested in studying the characteristics of projects within the ML Universe.

6.2. Data Analysis

The collected metrics were then used to evaluate whether a specific metric can be distinctly associated with a single ML category. Therefore, starting from the analyzed ML projects, we aimed to statistically analyze the differences between the categories.

Table 10: Metrics selected for analyzing community engagement, activity, and collaboration aspects of ML projects.

Metric Categories	
Engagement and Collaboration Metrics	**Number of Forks:** The number of times the repository has been forked by other users.
	Number of Stars: The number of stars the repository has received from users.
	Number of Contributors: The number of contributors to the repository.
Activity Metrics	**Number of Commits:** The number of commits of the repository.
	Activity Rate: The rate of activity measured as the frequency of commits over time.
Structural Metrics	**Project Age:** The amount of time passed since the creation date of the project.
	Size: The size of the repository expressed in lines of code (LOC).

Specifically, considering Table 10, let $X = \{\#Forks, \#Stars, \#Contributors, \#Commits, ActivityRate, ProjectAge, Size\}$ the set of metrics, we formulated the following null hypotheses for each of the proposed metrics:

H0: *For each $x \in X$, there is no statistically significant difference in terms of x between 'ML-Model Producers,' 'ML-Model Consumers,' 'ML-Model Producers & Consumers,' and 'ML Libraries & Toolkits.'*

This leads to the following alternative hypothesis, stating that for each dependent variable, a statistically relevant difference can be observed between categories:

Ha: *For each $x \in X$, there is a statistically significant difference in terms of x between 'ML-Model Producers,' 'ML-Model Consumers,' 'ML-Model Producers & Consumers,' and 'ML Libraries & Toolkits.'*

Before conducting any statistical tests, we first assessed the normality of our data to determine the appropriate statistical methods. We began by evaluating the properties of each distribution using the Shapiro-Wilk test [36] with a significance level of $\alpha = 0.05$, applied to each project category. This test helped us assess whether the data followed a normal distribution. If the data had been normally distributed, we would have employed parametric tests such as ANOVA. However, since the Shapiro-Wilk test revealed that the data did not conform to a normal distribution, we opted for non-parametric tests better suited to our data's characteristics. Specifically, we applied the Kruskal-Wallis H Test [28], a non-parametric method, to compare the medians across the four groups. This test is robust to the assumptions of normality homogeneity and provides an initial assessment of significant differences between the groups. Upon detecting significant differences using the Kruskal-Wallis H Test, we conducted pairwise Mann-Whitney U tests [48] with Holm-Bonferroni correction [38] to control for Type I errors. These pairwise comparisons allowed us to identify which specific ML project categories exhibit statistically significant differences from each other.

6.3. Analysis of the Results

Before analyzing the outcomes of the repository metrics analysis, we examined the prevalence of ML libraries in the project categories. Figure 3 shows the top 10 ML Libraries used respectively in *'ML-Model Producers'* and *'ML-Model Consumers.'* Common outcomes between ML project categories show the usage of common libraries, which is hugely different from other libraries. Specifically, "tensorflow" is the most used library for both the ML projects categories, 811 *'ML-Model Producers'* and 271 in *'ML-Model Consumers'* projects. This discrepancy between the most used libraries and others highlights the tendency of ML practitioners to use the most popular libraries, underrating the potentiality of other solutions. This phenomenon might be due "popularity bias" [15]. On the one hand, the main differences highlight the use of "sklearn" library in *'ML-Model Producer'* projects aimed specifically at producing ML models. On the other hand, "keras" library is more used than "sklearn" in *'ML-Model Consumer'* projects. Finally, other libraries that allow the training and use of different kind of ML models are highlighted in this analysis, such as "gym" library with 40 references in *'ML-Model Producer'* projects and "transformers" library, with 27 references in *'ML-Model Producer'* projects and nine in *'ML-Model Consumer'* Projects.

Hypothesis Testing. Several outcomes are enlighted when analyzing the community engagement, activity, and collaboration metrics. Table 11 shows the results of the Kruskal-Wallis H test applied for all the metrics. Additionally, results of Mann-Whitney U test post-hoc analysis are reported to identify statistically significant differences between ML project categories. Observing the results of the Friedman Test, all the metrics report a *p-value* lower than 0.005,
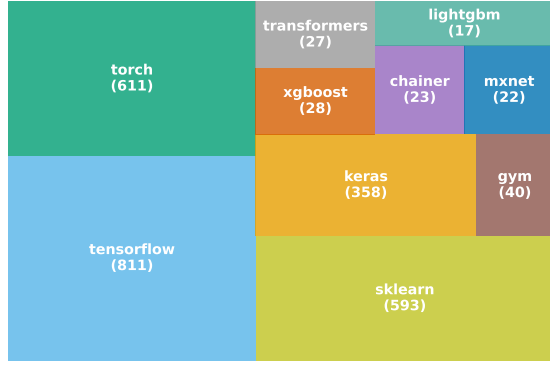
(a) Top 10 ML Libraries used in *'ML-Model Producers'* projects.

(b) Top 10 ML Libraries used in *'ML-Model Consumers'* projects.

Figure 3: Comparison of the top 10 ML libraries used by 'ML-Model Producers' and 'ML-Model Consumers'.

leading to the rejection of our formulated null hypotheses and concluding that there are significant differences between the three categories. The *number of stars* showed significant differences in several groups, such as {MP, LT} ($p < 0.001$) and {MPC, MP} ($p < 0.001$). In contrast, the difference between {MP, MC} (1) and {MPC, LT} (1) were not statistically significant. For the *size*, significant differences were observed in groups {MPC, MP} ($p < 0.001$), {MPC, MC} ($p = 0.03$), and {MPC, LT} ($p < 0.001$). However, groups such as {MC, LT} ($p = 1$) and {MP, MC} ($p = 1$) indicated there are no statistical differences. The *number of forks* followed a similar pattern, with significant differences between {MP, LT} ($p < 0.001$) and {MPC, MP} ($p < 0.001$). Nonetheless, the differences between {MP, MC} ($p = 0.74$) and {MPC, LT} ($p = 0.40$) and other groups were not statistically different. The *number of commits* exhibited significant differences in most groups such as {MP, LT} ($p < 0.001$) and {MPC, MP} ($p < 0.001$). However, the group {MP, MC} ($p = 0.87$) does not show significant statistical variations. For *project age*, significant differences were found in {MP, LT} ($p < 0.001$) and {MPC, LT} ($p < 0.001$). In contrast, the other groups, such as {MPC, MP} ($p = 1$) and {MPC, MC} ($p = 1$) demonstrated no statistical differences. The *activity rate* showed notable differences in most of the groups, such as {MP, LT} ($p < 0.001$) and {MPC, MP} ($p < 0.001$). However, the group {MP, MC} ($p = 0.82$), revealed no significant variation. Finally, the *number of contributors* exhibited significant differences in almost all groups, such as {MP, LT} ($p < 0.001$) and {MPC, MP} ($p < 0.001$). At the same time, there are no statistical differences in the group {MPC, MC} ($p = 0.82$). As a main outcome of the tests, the classification identified project categories with many characteristics for each group, each category of which could be associated with a repository metric. While many metrics displayed statistically significant differences across the groups, several comparisons revealed little or no variation, highlighting the diversity of certain project characteristics.

Table 11: Kruskal-Wallis H and Mann-Whitney U tests. Number(#), ML-Model Producer (MP), ML-Model Consumer (MC), ML-Library & Toolkit (LT), ML-Model Producers & Consumers (MPC).

Categories	Community engagement Metrics			Activity Metrics			Collaboration Metrics
Tests	# Stars	Size	# Forks	# Commits	Project Age	Activity Rate	Contributors
Kruskal-Wallis	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
Mann-Whitney {MP, MC}	1	1	0.74	0.87	1	0.82	0.008
Mann-Whitney {MP, LT}	<0.001	0.07	<0.001	<0.001	<0.001	<0.001	<0.001
Mann-Whitney {MC, LT}	0.02	1	0.1	<0.001	0.1	<0.001	0.003
Mann-Whitney {MPC, MP}	<0.001	<0.001	<0.001	<0.001	1	<0.001	<0.001
Mann-Whitney {MPC, MC}	0.02	0.03	0.03	<0.001	1	0.001	0.82
Mann-Whitney {MPC, LT}	1	<0.001	0.4	<0.001	<0.001	<0.001	<0.001

Statistical Analysis. To understand category differences, we observed their distribution, represented in Figure 4. MP and MC appear to have lower metric values than the two categories. Specifically, examining the distribution of stars across each ML project category, MP projects show fewer stars, with a median value of 93, compared to LT projects
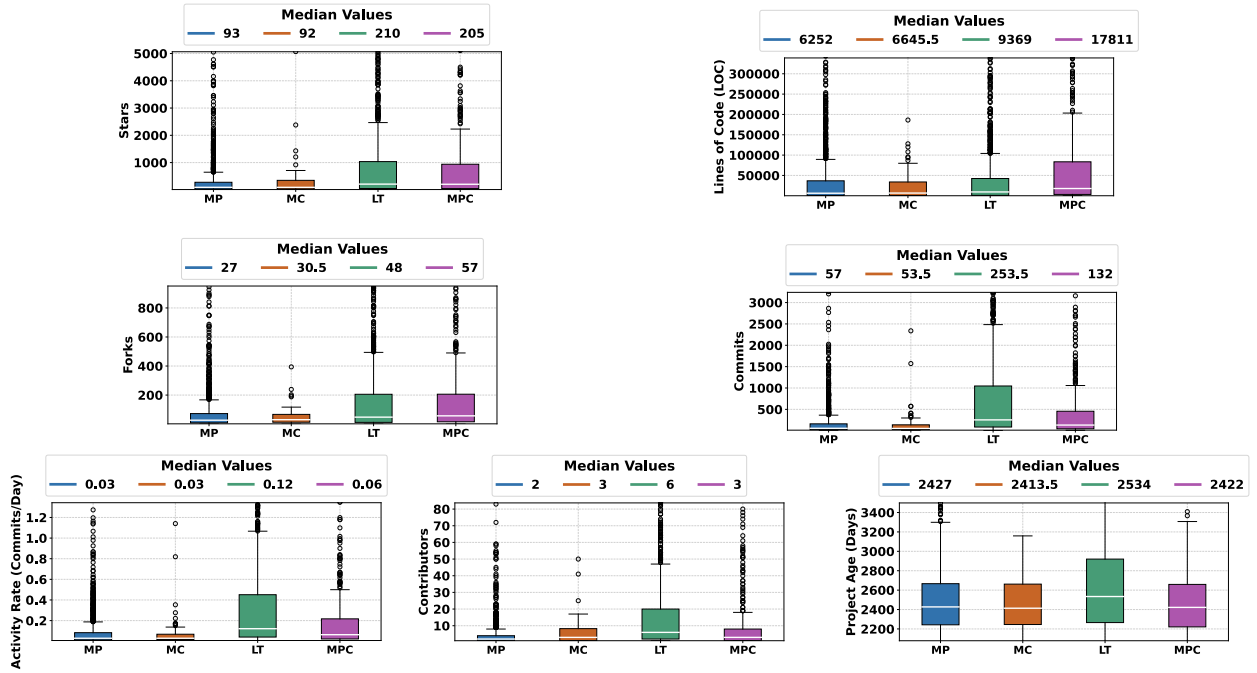
Figure 4: Distribution of the categories of ML projects among the selected metrics. The categories are ML-Model Producer (MP), ML-Model Consumer (MC), ML-Library & Toolkit (LT), ML-Model Producers & Consumers (MPC).

(210) and MPC projects (205). This difference was previously found to be statistically significant. In contrast, MC projects have a similar star distribution to MP, with a median value of 92. Overall, LT and MPC categories showed higher star counts compared to MP and MC. Statistically significant differences were also observed in project size, measured by LOC. MPC projects exhibited the largest size, with a median LOC of 17,811. LT projects followed with a median LOC of 9,369, notably higher than MP (median: 6,252) and MC (median: 6,645.5). Regarding the number of forks, MPC projects had the highest median value of 57. LT projects reported a slightly lower median of 48 forks. Meanwhile, the MP and MC categories showed fewer forks, with median values of 27 and 30.5, respectively. LT projects also reported the highest number of commits, with a median value of 253.5. In contrast, MPC projects had a lower median of 132 commits, while MP and MC categories displayed notably lower values, with medians of 57 and 53.5, respectively. The differences between MP and MC in this aspect are relatively small. A similar trend was observed for activity rate and the number of contributors, where LT and MPC projects generally outperformed MP and MC projects. Finally, regarding project age (measured as days since the starting date), LT projects reported the highest median value of 2,534 days. However, the differences between categories are minor, as the distributions are quite similar.

> ☝ **Answer to RQ₂.** Significant differences emerge across the project categories. Projects in the *'ML-Model Producers & Consumers'* and *'ML Library & Toolkit'* categories are characterized by larger environments, with higher median values for stars, size (LOC), forks, and commits. In contrast, *'ML-Model Producers'* projects are notably smaller and exhibit distinct differences, particularly in their lower median values across most metrics.

6.4. Threats to Validity

Given the nature of the projects analyzed, threats need to be considered during their characterization.

Internal Validity. One primary concern is the presence of confounding variables. Factors such as developer skill level could influence the selected metrics. We aimed to encompass metrics from various dimensions, ensuring a comprehensive analysis. The differences between the analyzed projects provide a valuable starting point for understanding the unique characteristics of ML projects.

23

Construct Validity. Regarding the definition of metrics used in the study, we acknowledge that metrics like the number of stars, forks, and commits are proxies for project quality and engagement. While these metrics might not fully capture a project's true impact or usability, they offer a practical and widely accepted method for evaluating project popularity and activity. For instance, the number of stars indicates a project's popularity [23], and the number of forks reflects the community's interest in contributing to the project [26]. Although these metrics might reflect surface-level popularity rather than deep engagement or technical excellence, they still provide valuable indicators of a project's visibility and activity.

Conclusion Validity. Many parametric tests are not applicable given the nature of our data distribution, which violates assumptions such as data independence between groups. Therefore, we applied the Kruskal-Wallis H Test [28] and Mann-Whitney U tests [48] with Holm-Bonferroni correction [38] to control for Type I errors. These tests are suitable for our data structure and ensure the robustness of our statistical analysis, providing reliable insights into the differences and relationships within ML-project categories.

7. Discussion and Implications

Our study highlights several implications for the SE4AI community, quantifying and shedding light on SE researchers' challenges and opportunities.

On the Value of MARK. RQ_1 shows that an automated classification is feasible and may reach good performance already with a heuristic approach, hence representing a valuable addition to the current state of the art [35, 73]. The automation of ML project classification addresses several key challenges in the field of SE4AI. One of the most critical issues is the substantial time and effort required for manual classification, especially when dealing with large datasets. Manual classification is a resource-intensive process that demands specialized domain knowledge and attention to detail, which increases the risk of errors and inconsistencies. These misclassifications can hinder the accuracy of subsequent analyses and limit the ability to draw meaningful conclusions from the data [24, 62]. By automating the classification process, our approach mitigates these challenges and enhances the quality and reliability of research outcomes. From a practical perspective, this automation empowers SE4AI researchers to conduct more precise and context-aware analyses of ML projects. For instance, by distinguishing among *'ML-Model Producer,'* *'ML-Model Consumer,'* and *'ML-Model Producer and Consumer,'* our classification framework allows researchers to tailor their studies to specific types of ML projects, leading to more relevant and actionable insights. For example, those focused on developing ML models might prioritize *'ML-Model Producers,'* those interested in applying and integrating pre-trained models could concentrate on *'ML-Model Consumers.'* In contrast, those aiming at exploring the full lifecycle of ML pipelines may focus on may focus on *'ML-Model Producer and Consumer.'* This represents a key transversal aspect that can significantly benefit the research community. By enabling comparative studies that analyze differences in how various types of ML projects operate—whether in their development processes, model deployment strategies, or maintenance practices—researchers can uncover critical insights that were previously overlooked. By accurately selecting and focusing on the right project samples, researchers can drive the creation of tailored best practices, tools, and methodologies specifically designed to meet the unique needs of each project type. This, in turn, will contribute to developing more effective and efficient ML systems across the board.

> 💡 **Implication #1.** Our classification approach has the potential to expand the scope of research in the field of SE4AI, possibly enabling new avenues of inquiry, such as more targeted studies on ML project types, refined tool development, and the creation of best practices tailored to specific project characteristics.

On the Challenges of Classifying ML Projects. Building on the findings of RQ_1, we argue that there are still some challenges for properly classifying ML projects. This is especially true in the case of *'ML-Model Consumer'*. Specifically, our study highlights the need to refine and expand the detection mechanism to cover a wider range of projects, finally improving its recall. In this respect, two considerations must be pointed out. Our assessment of various configurations of MARK revealed the intricate trade-offs between precision and recall. For example, Rule #3 significantly improved precision by minimizing false positives but reduced recall by excluding valid instances. This trade-off highlights the challenge of achieving a balance between these metrics. Nonetheless, the high precision achieved by the

final configuration of MARK is a significant asset for SE4AI research. Precision often holds greater importance in this domain than recall, as it ensures that the projects identified are reliable and accurate. By minimizing false positives, our classification provides a trustworthy foundation for subsequent research, enhancing the quality and reliability of findings and ensuring that the results can be confidently applied in practice. Similarly, our findings suggest that the final configuration of MARK strikes the best compromise between precision and recall while achieving the highest precision among all evaluated configurations. This optimal balance ensures that the approach aligns with its primary objective: providing a dependable tool for research that minimizes noise and misclassification, thereby enhancing the quality of downstream analyses. While improving recall remains a goal for future work, the current precision-focused approach may already effectively support research requiring high-quality datasets, where inclusivity is less critical than reliability.

At the same time, our analyses identified some limitations that warrant further research. In particular, a significant challenge lies in accurately distinguishing between projects that consume ML models and those that include similar functionalities for testing or auxiliary purposes. This issue often arises because current heuristics may not fully capture the contextual usage of APIs within the codebase. Two potential improvements can be considered to address this. First, adopting a *contextual analysis* approach: instead of relying solely on API calls, incorporating a deeper analysis of the files within the project—examining their purpose, interactions, and overall context—could significantly enhance classification accuracy. Second, leveraging *advanced AI-based solutions*, e.g., NLP techniques, could improve the analysis of comments, documentation, and code artifacts, thereby improving the accuracy of project classification. For instance, natural language processing techniques, such as pre-trained models like BERT or fine-tuned large language models, could be employed to disambiguate whether a file containing both producer and consumer APIs is used for evaluation or inference, hence contributing to increasing the overall classification accuracy of our approach. Similarly, prompt engineering could enable large language models to classify ML projects more effectively by leveraging context-aware insights from file descriptions. While promising, these approaches may require significant computational resources and further experimentation to assess their scalability and practicality.

> ♀ **Implication #2.** Further research is required to address the inherent challenges of classifying ML projects. Possible solutions involve contextual analysis and AI-based solutions to differentiate *'ML-Model Consumers'* from *'ML-Model Producers'*.

On the Use of the Classification in the Wild. When it turns to our second research question, we could first conclude that projects within the *'ML-Library & Toolkit,' 'ML-Model Producers,' 'ML-Model Consumers,'* and *'ML-Model Producers & Consumers'* categories exhibit significant differences in terms of repository metrics.

From a conceptual standpoint, the differences observed confirm the usefulness of the proposed classification and the overall significance of considering the nature of ML projects when performing empirical studies. Building on previous insights, the results underscore that differentiating ML projects is crucial for accurately investigating the structure, organization, and team dynamics of these projects. Similarly, our findings highlight the importance of tailored approaches when studying or developing tools for these projects. While our aim was not to exhaustively analyze all differences among project types, it is reasonable to expect that significant variations could also be observed in other areas, such as collaboration patterns, code quality metrics, or scalability concerns. As such, our findings represent a call to researchers in the field to explore these differences further, deepening our understanding of the unique challenges and opportunities presented by each type of ML project.

From a practical standpoint, our findings provide valuable insights into the socio-technical dynamics of ML projects, highlighting the distinct requirements and characteristics of different project categories. For instance, high levels of community engagement often correlate with strong support networks, essential for fostering innovation and sustaining project momentum. Similarly, activity levels within a project indicate its vitality and responsiveness to emerging developments, while patterns of contributor collaboration reflect the project's inclusivity and openness to diverse perspectives. The observed differences suggest varying maturity levels, impacting stability, reliability, and feature set comprehensiveness. Specifically, the analysis shows that different ML project categories have varying socio-technical requirements. *'ML-Library & Toolkit'* projects tend to have more commits and slightly more contributors, indicating a higher level of collaboration and a potentially more complex development process. These projects may benefit from robust communication and coordination tools to manage contributions effectively alongside more stringent quality assurance processes to maintain the integrity and performance of integrated models. Additionally,

'ML-Model Producers & Consumers' tend to exhibit higher lines of code and forks, reflecting their dual-purpose nature and the inherent complexity of combining model development with the deployment and distribution. The larger line of code suggests intricate pipelines and diverse functionalities, while the higher number of forks highlights their widespread use and adaptation by the community. These projects require technical expertise and collaborative practices to manage the full lifecycle of ML pipelines effectively. Conversely, 'ML-Model Producer' projects, with fewer contributors, may prioritize the depth of expertise and specialized skills in model development. In these cases, quality assurance may focus more on the accuracy and reliability of the models rather than broader collaborative processes. This differentiation is crucial for researchers when prioritizing efforts, as newer projects might be ideal for cutting-edge research. In comparison, older projects could be more suited for stable, long-term deployments.

The findings of our study may be instrumental for researchers in social software engineering, who are interested in understanding community dynamics and collaboration patterns, as well as for those working on software maintenance and evolution, who are focused on project longevity, stability, and the impact of maturity on software quality. In addition, given that socio-technical dynamics play a crucial role in shaping software quality [42, 56], the differences observed in our study suggest that these dynamics likely extend to the technical domain, potentially influencing the quality assurance practices employed across different ML projects. As such, our findings may stimulate additional research on the process and product quality of these systems.

> 💡 **Implication #3.** The differences observed in socio-technical dynamics are likely to influence other research domains, extending beyond the scope of our study. Our findings carry significant implications for researchers in areas like social software engineering, software maintenance and evolution, and software quality, providing them with valuable insights into the distinct characteristics of ML projects. These insights can be leveraged to understand better and address the unique challenges associated with these projects.

On Our Classification and Beyond. Our classification builds on existing knowledge of training and inference APIs, aiming not to be a static resource but an evolving effort that requires ongoing research and updates. The periodic updates and refinements of our knowledge base are fundamental for maintaining its relevance and usefulness over time. Our results show that even incremental updates to the KB can lead to modest but meaningful improvements in recall and F1-score, emphasizing the importance of periodically capturing the evolving landscape of ML tools and practices. This effort helps ensure that our classification remains robust and reflective of the current state of the field. We may keep the classification mechanism relevant and comprehensive by incorporating the latest advancements in ML APIs and analyzing emerging libraries that support cutting-edge models. Moreover, by making our study materials publicly available, we hope to foster collaboration within the research community. As periodic updates do not require substantial changes to the foundational rules and can focus on incremental refinements, this collaborative effort may ensure that MARK remains a scalable and long-lasting contribution to SE4AI research.

At the same time, we foresee future efforts extending our classification beyond the characterization of the currently supported classes of projects, i.e., 'ML-Model Producers,' 'ML-Model Consumers,' and 'ML-Model Producers and Consumers.' Although we did not systematically analyze the projects within each category, initial observations suggest differences that could lead to more refined classifications. Specifically, within 'ML-Model Producers' we identified variations in ML model architectures, ranging from traditional models like Decision Trees to more complex architectures such as CNNs. For instance, the MODEL-ZOO project [7] offers a variety of ML models, which the software engineering community could study to explore additional project categories and examine the differences in performance or non-functional requirements among different models. On the contrary, projects within 'ML-Model Consumer' tend to be more task-oriented, such as those focused on computer vision, audio, and NLP, which could be further studied in terms of their integration with other software components or their operation in more heterogeneous environments. Furthermore, our results show the massive adoption of libraries such as TensorFlow, Keras, Sklearn, and Torch for ML projects. This outcome suggests that studies that do not consider this set of libraries and frameworks can accidentally exclude a large number of ML projects, decreasing the generalizability of their results for researchers and practitioners. In addition, qualitative and quantitative studies could be leveraged to refine our classification further by exploring the underlying motivations for adopting certain popular libraries and frameworks over others. Qualitative approaches, e.g., surveys and interviews, could provide insights into developer preferences and decision-making processes. Meanwhile, quantitative methods, e.g., mining software repositories, could analyze usage

patterns and performance metrics across different projects. These studies could uncover trends and best practices, leading to context-aware classifications of the sub-types of projects within each category.

> ⚘ **Implication #4.** Our classification approach is the preliminary step for defining continuous evolving activities to identify and classify ML projects, enlarging opportunities for SE4AI research.

8. Conclusion

This study introduces the Mᴀᴄʜɪɴᴇ ʟᴇᴀʀɴɪɴɢ Aᴜᴛᴏᴍᴀᴛᴇᴅ Rᴜʟᴇ-ʙᴀꜱᴇᴅ Cʟᴀꜱꜱɪꜰɪᴄᴀᴛɪᴏɴ Kɪᴛ (MARK), a novel approach designed to classify Pʏᴛʜᴏɴ ML projects, distinguishing between *'ML-Model Producer,'* *'ML-Model Consumer,'* and *'ML-Model Producer & Consumer.'* Through empirical evaluation on 3,005 ML projects, MARK has shown a high F1-score, particularly for *'ML-Model Producer'* classifications, while some limitations should be still addressed for the classification of *'ML-Model Consumers.'* To sum up the contribution of this work, we provided:

1. MARK, a tool-supported classification approach able to categorize Pʏᴛʜᴏɴ ML projects;

2. A comprehensive knowledge base that can be utilized and expanded by researchers for further studies;

3. An empirical investigation into the effectiveness and potential usefulness of the proposed classification reveals that (1) MARK is accurate in most cases and (2) the classification made leads to the identification of significant differences among the classified ML projects;

4. An online replication package [18], to ensure the verifiability and reproducibility of our results.

As part of our future research agenda, we plan to explore the different socio-technical dynamics arising in ML projects of different natures. In addition, we plan to extend MARK knowledge base and improve its classification accuracy by exploring alternative methods, e.g., machine learning.

Acknowledgment

References

[1] [n. d.]. GitHub - ambakick/Person-Detection-and-Tracking: A tensorflow implementation with SSD model for person detection and Kalman Filtering combined for tracking — github.com. `https://github.com/ambakick/Person-Detection-and-Tracking`.

[2] [n. d.]. GitHub - bio-ontology-research-group/onto2vec: Representation learning for ontologies and their annotations — github.com. `https://github.com/bio-ontology-research-group/onto2vec`.

[3] [n. d.]. GitHub - bio-ontology-research-group/onto2vec: Representation learning for ontologies and their annotations — github.com. `https://github.com/bio-ontology-research-group/onto2vec`.

[4] [n. d.]. GitHub - daniel-cortez-stevenson/crypto-predict: A dockerized prediction API for crypto. — github.com. `https://github.com/daniel-cortez-stevenson/crypto-predict`.

[5] [n. d.]. GitHub - devpranoy/video_coloriser: Pytorch Convolutional Neural Net and GAN based video coloriser that converts black and white video to colorised video. — github.com. `https://github.com/devpranoy/video_coloriser`.

[6] [n. d.]. GitHub - IINemo/isanlp_srl_framebank: SRL parser for Russian based on FrameBank corpus — github.com. `https://github.com/IINemo/isanlp_srl_framebank`.

[7] [n. d.]. GitHub - openvinotoolkit/open_model_zoo: Pre-trained Deep Learning models and demos (high quality and extremely fast) — github.com. `https://github.com/openvinotoolkit/open_model_zoo`.

[8] [n. d.]. GitHub - sipeed/MaixPy-v1_scripts: micropython scripts for MaixPy — github.com. `https://github.com/sipeed/MaixPy-v1_scripts`.

[9] 2019. *cmasch/densenet*. `https://github.com/cmasch/densenet`

[10] 2019. GitHub - itsmehemant123/gpt2-discord-bot: discord bot, but its gpt-2 — github.com. `https://github.com/itsmehemant123/gpt2-discord-bot`.

[11] 2019. *imfunniee/pymine*. `https://github.com/imfunniee/pymine`

[12] 2022. GitHub REST API documentation - GitHub Docs — docs.github.com. `https://docs.github.com/en/rest`.

[13] 2023. *intuit/foremast-brain*. `https://github.com/intuit/foremast-brain`

[14] 2024. *bitextor/bitextor*. `https://github.com/bitextor/bitextor`

[15] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, and Bamshad Mobasher. 2019. The impact of popularity bias on fairness and calibration in recommendation. *arXiv preprint arXiv:1910.05755* (2019).

[16] Roozbeh Aghili, Heng Li, and Foutse Khomh. 2023. Studying the characteristics of AIOps projects on GitHub. *Empirical Software Engineering* 28, 6 (2023), 143.

[17] Nouh Alhindawi, Omar Meqdadi, Jamal Alsakran, N Aljawarneh, and H Migdadi. 2022. Understanding and predicting bugs fixed by API-migrations. *International Journal of Data and Network Science* 6, 3 (2022), 849–860.

[18] A. Anonymous. 2023. *Into the ML-universe: An Improved Classification and Characterization of Machine-Learning Projects. Online Appendix.* `https://doi.org/10.6084/m9.figshare.25974886`

[19] Sebastian Baltes and Paul Ralph. 2022. Sampling in software engineering research: A critical review and guidelines. *Empirical Software Engineering* 27, 4 (2022), 94.

[20] P Bengtsson and Jan Bosch. 1999. Architecture level prediction of software maintenance. In *Proceedings of the Third European Conference on Software Maintenance and Reengineering (Cat. No. PR00090)*. IEEE, 139–147.

[21] João Helis Bernardo, Daniel Alencar da Costa, Sérgio Queiroz de Medeiros, and Uirá Kulesza. 2024. How do Machine Learning Projects use Continuous Integration Practices? An Empirical Study on GitHub Actions. *arXiv preprint arXiv:2403.09547* (2024).

[22] Sumon Biswas, Md Johirul Islam, Yijia Huang, and Hridesh Rajan. 2019. Boa Meets Python: A Boa Dataset of Data Science Software in Python Language. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 577–581. `https://doi.org/10.1109/MSR.2019.00086`

[23] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 334–344.

[24] Fabio Calefato, Filippo Lanubile, and Luigi Quaranta. 2022. A preliminary investigation of MLOps practices in GitHub. In *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 283–288.

[25] Gemma Catolino, Fabio Palomba, Andy Zaidman, and Filomena Ferrucci. 2019. Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software* 152 (2019), 165–181.

[26] Fangwei Chen, Lei Li, Jing Jiang, and Li Zhang. 2014. Predicting the number of forks for open source software project. In *Proceedings of the 2014 3rd International workshop on evidential assessment of software technologies*. 40–47.

[27] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[28] William Jay Conover. 1999. *Practical nonparametric statistics*. Vol. 350. john wiley & sons.

[29] Fastai Contributors. 2024. GitHub - fastai/fastai: The fastai deep learning library — github.com. `https://github.com/fastai/fastai`.

[30] Valerio Cosentino, Javier L Cánovas Izquierdo, and Jordi Cabot. 2017. A systematic mapping study of software development with GitHub. *Ieee access* 5 (2017), 7173–7192.

[31] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. 1277–1286.

[32] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling projects in github for MSR studies. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 560–564.

[33] Elizamary de Souza Nascimento, Iftekhar Ahmed, Edson Oliveira, Márcio Piedade Palheta, Igor Steinmacher, and Tayana Conte. 2019. Understanding development process of machine learning systems: Challenges and solutions. In *2019 acm/ieee international symposium on empirical software engineering and measurement (esem)*. IEEE, 1–6.

[34] Wei Fu and Tim Menzies. 2017. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 49–60.

[35] Danielle Gonzalez, Thomas Zimmermann, and Nachiappan Nagappan. 2020. The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on github. In *Proceedings of the 17th International conference on mining software repositories*. 431–442.

[36] Elizabeth González-Estrada and Waldenia Cosmes. 2019. Shapiro–Wilk test for skew normal distributions based on data transformations. *Journal of Statistical Computation and Simulation* 89, 17 (2019), 3258–3272.

[37] Georgios Gousios and Diomidis Spinellis. 2017. Mining software engineering data from GitHub. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 501–502.

[38] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[39] Geoff Hulten. 2018. *Building Intelligent Systems: A Guide to Machine Learning Engineering*. Apress.

[40] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21 (2016), 2035–2071.

[41] Amela Karahasanovic, Erik Gausta Nilsson, Giorgio Grani, Kjell Fredrik Pettersen, Leo Karabeg, and Patrick Schittekat. 2021. User Involvement in the Design of ML-Infused Systems. In *CHI Greece 2021: 1st International Conference of the ACM Greek SIGCHI Chapter*. 1–5.

[42] Irwin Kwan, Adrian Schroter, and Daniela Damian. 2011. Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE Transactions on Software Engineering* 37, 3 (2011), 307–324.

[43] Valliappa Lakshmanan, Sara Robinson, and Michael Munn. 2020. *Machine learning design patterns*. O'Reilly Media.

[44] Grace A Lewis, Ipek Ozkaya, and Xiwei Xu. 2021. Software architecture challenges for ml systems. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 634–638.

[45] Hao Li and Cor-Paul Bezemer. 2022. Studying Popular Open Source Machine Learning Libraries and Their Cross-Ecosystem Bindings. arXiv:2201.07201 [cs.SE]

[46] Xiaoyu Liu, LiGuo Huang, and Vincent Ng. 2018. Effective API recommendation without historical software repositories. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 282–292.

[47] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2022. Software engineering for AI-based systems: a survey. *ACM Transactions on Software Engineering and Methodology (TOSEM)*

31, 2 (2022), 1–59.

[48] Patrick E McKnight and Julius Najab. 2010. Mann-Whitney U Test. *The Corsini encyclopedia of psychology* (2010), 1–1.

[49] Collin McMillan, Mario Linares-Vasquez, Denys Poshyvanyk, and Mark Grechanik. 2011. Categorizing software applications for maintenance. In *2011 27th ieee international conference on software maintenance (icsm)*. IEEE, 343–352.

[50] Claire Cain Miller. 2015. Can an algorithm hire better than a human. *The New York Times* 25 (2015).

[51] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating github for engineered software projects. *Empirical Software Engineering* 22 (2017), 3219–3253.

[52] Nadia Nahar, Shurui Zhou, Grace Lewis, and Christian Kästner. 2022. Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process. In *Proceedings of the 44th international conference on software engineering*. 413–425.

[53] Giang Nguyen, Stefan Dlugolinsky, Martin Bobák, Viet Tran, Álvaro López García, Ignacio Heredia, Peter Malík, and Ladislav Hluchỳ. 2019. Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review* 52 (2019), 77–124.

[54] Parmy Olson. 2011. The algorithm that beats your bank manager. *CNN Money March* 15 (2011).

[55] Abbas Ourmazd. 2020. Science in the age of machine learning. *Nature Reviews Physics* 2, 7 (2020), 342–343.

[56] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. 2018. Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE transactions on software engineering* 47, 1 (2018), 108–129.

[57] Federica Pepe, Fiorella Zampetti, Antonio Mastropaolo, Gabriele Bavota, and Massimiliano Di Penta. 2024. A Taxonomy of Self-Admitted Technical Debt in Deep Learning Systems. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 388–399.

[58] David MW Powers. 2020. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061* (2020).

[59] Sebastian Raschka, Joshua Patterson, and Corey Nolet. 2020. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information* 11, 4 (2020), 193.

[60] Jörg Rech and Klaus-Dieter Althoff. 2004. Artificial intelligence and software engineering: Status and future trends. *KI* 18, 3 (2004), 5–11.

[61] Harvard Business Review. 2023. AI Is Not Just Getting Better, it's Becoming More Pervasive. `https://hbr.org/sponsored/2019/02/ai-is-not-just-getting-better-its-becoming-more-pervasive`.

[62] Dhia Elhaq Rzig, Foyzul Hassan, Chetan Bansal, and Nachiappan Nagappan. 2022. Characterizing the Usage of CI Tools in ML Projects. In *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 69–79.

[63] Santonu Sarkar, Girish Maskeri Rama, and Avinash C Kak. 2006. API-based and information-theoretic metrics for measuring the quality of software modularization. *IEEE Transactions on Software Engineering* 33, 1 (2006), 14–32.

[64] Julien Siebert, Lisa Joeckel, Jens Heidrich, Koji Nakamichi, Kyoko Ohashi, Isao Namba, Rieko Yamamoto, and Mikio Aoyama. 2020. Towards guidelines for assessing qualities of machine learning systems. In *Quality of Information and Communications Technology: 13th International Conference, QUATIC 2020, Faro, Portugal, September 9–11, 2020, Proceedings 13*. Springer, 17–31.

[65] TU/e Robotics Team. 2024. GitHub - tue-robotics/image_recognition: Packages for image recognition - Robocup TU/e Robotics — github.com. `https://github.com/tue-robotics/image_recognition`.

[66] Steven K Thompson. 2012. *Sampling*. Vol. 755. John Wiley & Sons.

[67] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. 2009. Using latent dirichlet allocation for automatic categorization of software. In *2009 6th IEEE international working conference on mining software repositories*. IEEE, 163–166.

[68] Gias Uddin and Martin P Robillard. 2015. How API documentation fails. *Ieee software* 32, 4 (2015), 68–75.

[69] Secil Ugurel, Robert Krovetz, and C Lee Giles. 2002. What's the code? automatic classification of source code archives. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. 632–638.

[70] Melina Vidoni. 2022. A systematic process for Mining Software Repositories: Results from a systematic literature review. *Information and Software Technology* 144 (2022), 106791.

[71] Hugo Villamizar, Marcos Kalinowski, and Hélio Lopes. 2022. Towards perspective-based specification of machine learning-enabled systems. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 112–115.

[72] Chengcheng Wan, Shicheng Liu, Henry Hoffmann, Michael Maire, and Shan Lu. 2021. Are machine learning cloud apis used correctly?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 125–137.

[73] Ratnadira Widyasari, Zhou Yang, Ferdian Thung, Sheng Qin Sim, Fiona Wee, Camellia Lok, Jack Phan, Haodi Qi, Constance Tan, Qijin Tay, et al. 2023. NICHE: A Curated Dataset of Engineered Machine Learning Projects in Python. *arXiv preprint arXiv:2303.06286* (2023).

[74] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[75] Hao Zhong and Zhendong Su. 2013. Detecting API documentation errors. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*. 803–816.

[76] Jianlong Zhou and Fang Chen. 2018. *Human and Machine Learning*. Springer.