

A Large-Scale Study of Model Integration in ML-Enabled Software Systems

Yorick Sens*, Henriette Knopp*, Sven Peldszus*, Thorsten Berger*,†
*Ruhr University Bochum, Germany †Chalmers | University of Gothenburg, Sweden

2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE) | 979-8-3315-0569-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICSE55347.2025.00185

***Abstract*—The rise of machine learning (ML) and its integration into software systems has drastically changed development practices. While software engineering traditionally focused on manually created code artifacts with dedicated processes and architectures, ML-enabled systems require additional data-science methods and tools to create ML artifacts—especially ML models and training data. However, integrating models into systems, and managing the many different artifacts involved, is far from trivial. ML-enabled systems can easily have multiple ML models that interact with each other and with traditional code in intricate ways. Unfortunately, while challenges and practices of building ML-enabled systems have been studied, little is known about the characteristics of real-world ML-enabled systems beyond isolated examples. Improving engineering processes and architectures for ML-enabled systems requires improving the empirical understanding of these systems.**

We present a large-scale study of 2,928 open-source ML-enabled software systems. We classified and analyzed them to determine system characteristics, model and code reuse practices, and architectural aspects of integrating ML models. Our findings show that these systems still mainly consist of traditional source code, and that ML model reuse through code duplication or pre-trained models is common. We also identified different ML integration patterns and related implementation practices. We hope that our results help improve practices for integrating ML models, bringing data science and software engineering closer together.

***Index Terms*—machine learning, AI engineering, SE4AI**

I. Introduction

Many recent breakthroughs in machine learning (ML) have given rise to ML-enabled software that was not realizable before. Consider cyber-physical systems, such as autonomous vehicles [1], [2] or unmanned aerial vehicles [3], as well as software in finance [4] or healthcare [5]. All these systems benefit from advances in ML model architectures and training algorithms. Nevertheless, developers still struggle when developing software systems that integrate ML models [6], [7]. In fact, industrial surveys report that 78 % of ML projects stall [8], while others say as many as 85 % fail [9]. Many factors can cause such failures [10], including development processes [11], [12], [13], data quality [14], system architecture [15], tool support [16], [17], [18], [19], [20], [21], quality assurance [22], and the integration of ML models with traditional software components [23], [24]—the focus of our work.

Integrating ML models in software systems is challenging. Executing the models requires boilerplate code and extra infrastructure [20]. The lack of behavioral guarantees in ML models—being probabilistic by nature—requires additional safeguards, especially in safety-critical systems [22]. Safeguards are typically implemented manually to prevent undefined behavior for certain inputs, as well as invalid results that would violate domain restrictions. These challenges are further exacerbated by systems that integrate multiple models. Extreme examples are perception systems in autonomous driving, some of which boast up to 28 different ML models that interact with each other [25] and are safeguarded with manually implemented heuristics. Furthermore, training and developing ML models is often costly [26], [27], which made ML model reuse a common practice, In particular, reusing pre-trained ML models allows leveraging larger models trained on more data, but does not fit well with current software engineering practices. Unfortunately, empirical insights on reuse practices are missing.

Research on engineering ML-enabled systems has focused on challenges and practices of engineering such systems [11], [14], [22], [13], [18], [23]. Researchers also created novel workflows and tooling (e.g., experiment management tools [17], [28]) to address the challenges. Much work also focused on dependability and safety of ML models [29]. However, building ML-enabled systems requires holistic methods and tools—to systematically integrating traditional software and data-science artifacts in a whole system, as well as maintaining and evolving such systems. Progress in this area has been limited so far, which can be attributed to a lack of understanding of the current practices and concrete problems of developing ML-enabled software systems. In other words, while researchers have focused on ML engineering, still, little is known about how ML models are integrated into ML-enabled systems in practice.

We present an empirical study of 2,928 ML-enabled software repositories on GitHub, characterizing the use of ML in software systems. We focused on the reuse of ML models across software systems, as well as the integration of ML models. Our analysis addresses three main research questions:

RQ1*: *What are characteristics of ML-enabled software systems? To understand such systems, we studied what role ML plays in them and what characterizes them. For instance, what is the proportion of ML code in relation to non-ML source code, what types of software systems employ ML (e.g., libraries or end-user-oriented applications), and how?

RQ2*: *How are ML models reused across software systems? The growing size (e.g., number of trainable parameters) of ML models both enables and necessitates their reuse across software systems—as they become more general in their capabilities, but also more difficult to train [30], [31]. We explored how reuse is done in practice, like reuse of source code has been studied in previous works [32], [33].

RQ3*: *How is the integration of ML models reflected in architectural aspects of software systems? We explored how

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00185

ML models are integrated into systems and what functionality they realize. We identified how many models are used, how they interact, and how they are integrated with code, for instance, with pre- and post-processing methods.

We contribute:

- insights on the **share of ML-related code** of ML-enabled systems and its distribution, showing that even **in ML-enabled systems traditional source code makes up for most parts of the system**;
- insights into **ML reuse practices**, highlighting a **strong reliance on pre-trained models**, as well as the practice of **copying source code implementing ML models** between systems;
- a **catalog of ML integration patterns** and coding practices derived from an **in-depth analysis of 26 applications** that use ML to provide end-user-oriented functionality;
- and a **dataset of 2,928 ML-enabled software systems**, of which we manually classified 160 according to the type of the system and their relation to ML.

For replication and further studies, we provide a replication package, including the dataset and analysis scripts [34]. We hope that our findings will help practitioners and researchers better understand the integration of ML models in ML-enabled systems, and that our data will be useful for further research.

II. Motivation and Research Questions

We present the necessary background, the state of research, as well as motivate and refine our research questions.

ML-enabled systems contain ML-based functionality, ranging from utility functions to application logic. This functionality can include predictions, data analysis, or generative content creation [35]. ML-enabled systems incorporate various ML technologies, such as Deep Learning (DL), Convolutional Neural Networks (CNNs), or Transformers [36], [37], [38]. This integration of ML into software *poses new challenges for development processes, architectures, and testing, among others* [14]. Many originate from the fundamentally different nature of ML models compared to traditional software [24]. ML models are probabilistic, essentially constituting unreliable functions, while traditional software is more deterministic.

Characteristics of ML-Enabled Software Systems (RQ1). While there exists extensive theory in data science, the majority of techniques is model-centric. However, integrating ML models into software systems to provide value to end-users requires system-centric engineering methods. Effective methods need to be tailored towards the actual characteristics and properties of ML in the context of systems. To this end, we need to improve our empirical understanding of ML-enabled systems in general. Furthermore, software systems can be of different types, such as applications, libraries, and frameworks [39], [40], which impacts the use of ML. ***RQ1.1*** *(What types of systems use ML and how are they related to it?)* first determines what systems [39], [40] (e.g., applications or libraries) integrate ML models and what ML is used for (e.g., for end-user functionality or purely conceptual contributions). This puts the results of our study into context and helps identify differences in the use of ML, as well as implications for practice.

ML-enabled systems rely on ML libraries and frameworks. The most popular ones [41] are PyTorch [42], TensorFlow [43], and Scikit-Learn [44]. Many others build upon them. While TensorFlow and PyTorch offer DL, Scikit-Learn offers different techniques, mostly traditional ML ranging from supervised methods (e.g., linear regression and perceptron) to unsupervised methods (e.g., clustering and PCA). TensorFlow and PyTorch are frameworks designed for a modular implementation of ML models. A class "Module" (*"BaseEstimator"* in Scikit-Learn) represents either complete models or their building blocks (e.g., layers). Developers typically extend this class, often using predefined modules, such as linear or convolution layers.

ML-enabled systems contain additional artifacts (called *ML assets* [17] in the remainder), including model implementations, model binaries, ML training code, and other ML files. These ML assets must be integrated with code and co-evolved. ***RQ1.2*** *(How much of ML-enabled systems is specific to ML?)* elicits details on the relevance, type, and characteristics of ML assets.

Finally, while quality assurance is an integral part of ML model training, also the system has to be considered. However, the concrete quality assurance practices in ML-enabled systems have not been captured on a large scale, beyond individual qualitative studies [45]. To obtain insights on these practices, ***RQ1.3*** *(What are quality assurance practices of ML-enabled systems?)* investigates quality assurance on a large scale.

Reuse of ML Models (RQ2). Like reuse in traditional software [33], developers reuse ML assets. It is commonly known that there is reuse of established ML models, allowing developers to use them without the extensive effort required to build a new model [31], [30]. Such models are called pre-trained models, as they are optimized and fully functional. Pre-trained models for specific tasks are distributed through model hubs. Jiang et al. [46] identified 8 model hubs when studying the content of their provided artifacts and related security risks: PyTorch Hub [47], Tensorflow Hub [43], Hugging Face [48], Model Zoo [49], ONNX Model Zoo [50], Model Hub [51], NVIDIA NGC [52], and MATLAB Model Hub [53]. While we know all these sources from which pre-trained models could be reused, we know little about the actual practices, neither how prevalent this reuse is nor how it is realized. ***RQ2.1*** *(To what extent are pre-trained models used?)* addresses this gap.

Apart from full models, other ML assets (e.g., scripts, model binaries) can be reused as well. However, there are currently no studies on such reuse and its relation to model reuse, e.g., reusing implementations of ML models as an alternative to pre-trained models. Related studies focus on how to identify model plagiarism [54], or on reuse practices of models from specific libraries or platforms [55]. Consequently, we need to improve our empirical understanding to what degree ML models are currently reused, what other assets are also reused, and how that reuse is realized—addressed by ***RQ2.2*** *(To what extent are ML assets reused between ML-enabled systems?).*

Architectural Aspects (RQ3). The architecture of ML-enabled systems revolves around ML models, ML assets, and traditional

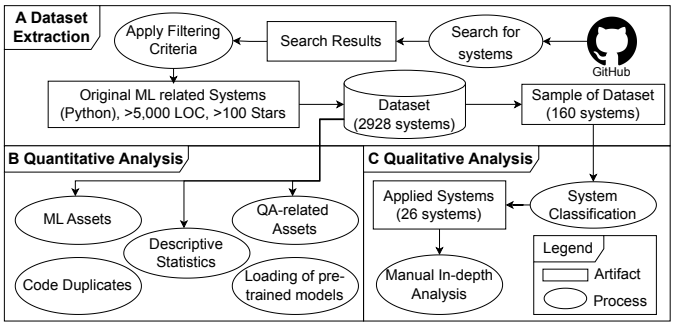

Fig. 1: Methodology overview

software, which must be integrated together. However, little is known on how the interaction of ML models can impact the behavior of a system. While some architectures for ML-enabled systems have been proposed, they usually do not focus on concrete interactions and topologies of ML models, but rather on high-level system abstractions. This includes, for instance, a microservice architecture, where ML models are modularized and interact with the rest of the system via REST [56]. Others suggest separating ML logic from the business logic of a system [57].

ML models are typically encapsulated in software modules that contain the ML model (binary) along with source code for model loading and execution, including required pre- and post-processing steps. Common anti-patterns include excessive glue code, complicated data-processing pipelines, dead experimental code paths, and lack of abstraction [20]. Since ML models are updated or replaced regularly [58], i.e., by an optimized version, which in turn affects the remaining system (e.g., thresholds in glue code need to be adjusted), such anti-patterns hinder optimization. Little is known about best practices in such situations. To derive such, ***RQ3.1*** *(How are ML models embedded into traditional code?)* investigates current ML model integration patterns.

Complex systems can integrate many ML models. Recall perception systems with up to 28 models, integrated in complex topologies, with non-trivial interactions and dependencies between models [25]. While current research discusses the implications of such topologies (e.g., propagating error, computational complexity) [59], [60], it is important to study these in a larger dataset and define common patterns—addressed by ***RQ3.2*** *(What are interaction patterns of ML models?)*.

III. Methodology

Our methodology, summarized in Fig. 1, comprised the mining of repositories with ML-enabled systems from GitHub, and their quantitative and qualitative analysis. While some sub-RQs could be answered by analyzing the whole dataset automatically, other sub-RQs required qualitative analysis of random samples.

A. Subject Selection

We mined repositories from GitHub as follows, considering typical mining strategies [61], [62].

Inclusion Criteria. We used GitHub's dependency graph to obtain repositories that use one of the three most popular ML libraries [41] (IC1) and that are implemented in the currently most popular programming language for ML (Python) (IC2).

IC1 *The repository depends on TensorFlow [43], PyTorch [42], or scikit-learn [44].* This criterion aimed to identify systems implementing or using ML technologies. ML libraries support developing (especially training) ML models, but are also needed to execute the models.

IC2 *The main programming language is Python.* This criteria aimed to ensure the relevance of our analysis—since Python is the currently most popular language for ML-enabled systems [63], [64]—and the comparability of the subject systems by focusing only on one language.

Applying IC1 yielded 754,161 repositories, of which 255,096 dependent on PyTorch, 233,779 on TensorFlow, and 465,980 on Scikit-Learn, and IC2 in 405,091 Python repositories.

Exclusion Criteria. We then filtered the obtained repositories to create a high-quality set of ML-enabled software systems of a certain size and maturity. We filtered out repositories if at least one of the following criteria was met.

EC1 *The repository is not original, i.e., forked or duplicated from another repository in the dataset.* Forks would distort the results, as popular projects with many forks would be counted multiple times and disproportionately affect the results, without providing additional insights.

EC2 *The repository has fewer than 100 stars.* Sufficient popularity ensures to avoid practically irrelevant projects.

EC3 *The codebase contains fewer than 5,000 lines of Python code.* This criterion aimed at ensuring a certain size of the code, excluding tutorials and toy projects.

The thresholds for EC2 and EC3 were chosen based on previous studies [64], [65], [40], [66], [67] and manual exploration of a sample. We set them relatively low to exclude simple toy projects, but still obtain a large, representative dataset.

We excluded forks with EC1 in two steps: First, we removed projects that were forked directly from GitHub, which can be retrieved through GitHub's REST API. Second, since we found that the dataset still contained a significant number of duplicates, we identified them based on identical README files and kept only the repository that was created first. This selection resulted in a final dataset of 2,928 repositories.

B. Quantitative Analysis

We analyzed the ML-enabled systems by creating analysis scripts and using automated tools. That was possible for the following research questions, which we answered as follows.

Descriptive Statistics. To put all our results into context—and to determine to what kinds of systems they apply—we first created descriptive statistics. Typical descriptive statistics are scale and popularity of the subjects [68]. For popularity, we used GitHub Stars, which can also be seen as an indicator of maturity. To determine the systems' scales, we used common code metrics [68]: number of commits, source files, lines of code, number of classes and functions. These metrics help to contextualize results and highlight the scale and relevance of our dataset. We also provide statistics on the use of ML libraries, also whether multiple libraries are used.

General Characteristics (*RQ1*). The first research question, that can be answered quantitatively on the full dataset, is ***RQ1.2*** *(How much of ML-enabled systems is specific to ML?)*, since we were able to automatically identify ML assets, use metrics to measure their characteristics, and compare those to traditional software assets. To identify the ML-related parts of our subject systems, we extracted ML assets by searching for the following typical ML assets:

Implementations of ML models: Recall that models are implemented by extending the class *"Module"* (PyTorch and TensorFlow) or *"BaseEstimator"* (Scikit-Learn) (Sec. II). We searched for class definitions that inherit from these base classes or any of their subclasses provided in the libraries or defined in the subject systems.

ML-related functions: We extracted all functions and class methods that either use an API of an ML library or instantiate an ML model. Functions defined within an ML model are considered ML-related, regardless of whether the specific function directly interacts with an ML library.

Stored model files: ML models are often stored in binary files for later reuse, typically using the extensions .pt or .pth (PyTorch) and .h5 or .pb. (TensorFlow). We counted the number of such files present in the systems.

We compared the general code metrics of the traditional code assets to those of the identified ML assets. This analysis provides insights into the extent of the ML-specific assets compared to traditional software assets.

We also answered ***RQ1.3*** *(What are quality assurance practices of ML-enabled systems?)* quantitatively on our full dataset, using automated techniques. We analyzed to what degree the source code, especially the ML-related part, is covered by unit tests, and how the models are evaluated. To identify test cases, we searched the source code for implementations of Python's unittest framework. We then identified the unit under test by analyzing the method calls invoked. This allowed us to measure the number of functions directly covered by tests for ML-and non-ML functions, respectively. To investigate quality assurance of ML models, we extracted usages of ML validation functions, following another work analyzing ML-related projects [39].

Reuse of ML Models (*RQ2*). The research question ***RQ2.1*** *(To what extent are pre-trained models used?)* was answered quantitatively on the full dataset, since we were able to detect the presence of pre-trained models automatically. Recall that using pre-trained models distributed through model hubs (Sec. II) is a systematic way of model reuse. We relied on the list of Jiang et al. [46]. From each hub's documentation we extract a list of APIs used to load pre-trained models in Python. The full list can be found in our replication package [34]. We then identified which systems use the APIs and where the model is loaded in the system, collecting a list of all related files. Furthermore, we analyzed the source of the API calls in the system by categorizing them as test, demonstration, and other directories, according to their name. We report the number of systems that contain an API used to load pre-trained models and where they are loaded.

We also analyzed reuse of ML assets at a large scale among all systems in our full dataset, answering ***RQ2.2:*** *(To what extent are ML assets duplicated between ML-enabled systems?)* quantitatively. Specifically, to identify duplicated ML code, we applied a code clone detection tool to the ML-enabled systems. We decided to use JPlag [69], which supports Python and is ranked as one of the best tools in multiple literature reviews [70], [71], [72]. In a pairwise comparison, a match is found when a sequence of tokens of a certain minimum length for two files match. We used a threshold of 100 tokens to avoid false positives. Based on these results, we identified the most prominent sources of duplicated model implementations by sorting the subject systems by the number of duplicated tokens shared with all others systems. Then, starting from the projects with the highest amount of duplicated code, we manually inspected each system, to determine if they are original or what system they are based on. To this end, we analyzed the copied source files and searched for hints on the original system. Usually, the copyright information was left in place, crediting the original repository. This analysis was repeated until we encountered no more original systems for 10 times in a row.

We report the number of code duplicates from each original system in our dataset. Since some original systems were not contained in the dataset (e.g. because they were not stored on GitHub), we use another system, that only contains a complete duplicate from this system and no other, as a placeholder to report the number of duplicates from the original one.

C. Qualitative Analysis: System Classification

For our qualitative analysis, we selected a random sample of 160 systems (5.5 % of our dataset) and answered the following research questions requiring qualitative analysis.

To answer ***RQ1.1*** *(What types of systems use ML and how are they related to it?)* we classify systems inspired by studies [39], [64], [73], [21] that investigated properties of ML-enabled systems, such as ML model development stages, and quality assurance. More specifically, we classify according to the following two dimensions: *type* of systems (i.e., Application or Library) and their *relation to ML* (i.e., what are the ML based functionalities and how are they used).

We manually examined and labeled the sample in several iterations, in which the definitions of system type and relation to ML were continuously adjusted. Two authors classified batches of systems independently and then discussed their disagreements. A third author was the mediator if no consensus was reached. We started with 100 projects to develop an understanding of what is present in the dataset and to define the labels. Thereafter, the two authors classified batches of 15 systems independently. We used Krippendorff's Alpha [74], [75] for the agreement score. As the score was below 0.6 in the first iterations, the authors discussed their disagreements. After four iterations we achieved an alpha of 0.86 for the system type and an alpha of 0.70 for the relation of ML, confirming a common knowledge base and correctness of the assigned labels. Based on the improved understanding, the previous systems (including the initial sample) were reclassified by one author, with the second author checking the results.

D. Qualitative Analysis: Manual In-Depth Analysis

To answer ***RQ3*** and its sub-research questions ***RQ3.1*** *(How are ML models embedded into traditional code?)* and ***RQ3.2*** *(What are interaction patterns of ML models?)* a manual analysis of the source code was necessary to identify relations of ML models and traditional software components. Since analyzing 160 systems manually is not feasible we selected all systems that are end-user-oriented applications and that use ML to realize business logic (type "Application" and label "Business-Focused Applied," cf. Sec. III-B). For the selected systems, we collect insights on architectural patterns and report concrete details through examples. Our analysis is inspired by Peng et al. [25], who studied various aspects of model integration in an autonomous driving perception system (cf. Sec. II). Upon the research questions and our experience, we defined relevant aspects to extract. We then systematically examined the implementation of each system and documented observations related to the aspects (the raw data is in our appendix [34]). We time-boxed the manual analysis to two hours per system. If we could not gather enough information in time, for instance, due to insufficient documentation or poorly structured source code, we noted the respective aspect as unknown.

To structure the analysis, we first identified the code related to the ML models, as described in Sec. III-B, as entry points for manual exploration. For each entry we analyzed the surrounding code to understand the model integration. Then, using the IDE's call-graph navigation, we determined where models are instantiated. We disregarded instantiations that are tests or demonstrations, or that occur in files that are no longer used but are kept by developers. For instantiations that occur in actual production code, we then analyzed the surrounding code to determine the input and output of the model. If necessary, we followed the method calls further to clarify the data format of the input and output, as well as possible pre- and post-processing steps, characterizing the data flow and topologies of the models in a system. After checking each instantiation for each model, we summarized the results to obtain the total number of models used. All projects are details in our online appendix [34].

We collected the following aspects. The former are descriptive aspects putting the results from the quantitative analysis (III-B) into context. The latter are related to model integration, answering ***RQ3.1***, and interaction patterns, answering ***RQ3.2***.

Descriptive Aspects. While these aspects are related to general characteristics of ML-enabled systems (***RQ1***) and practices on model reuse (***RQ2***), our qualitative analysis goes into much more technical detail on the sample of 26 systems.

ML Functionalities and Tasks: *What are the ML models used for? What functionality is implemented with ML? Based on which ML task is the functionality implemented?*

Number and Type of Models: *How many ML models are used? What is their model architecture (CNN, RNN, etc)? How are they trained (supervised, unsupervised, etc.)?*

Model Origins: *Is the model custom-built, completely pre-trained or pre-trained with additional fine-tuning?* Besides quantitative statistics about model reuse (***RQ2***), we complemented these with qualitative aspects (e.g., how tailored).

Model Embedding. The following aspects address ***RQ3.1*** *(How are ML models embedded into traditional code?)*

Storing & Loading of Models: *Are the models stored locally in the repository or loaded from a remote source, and how?* Models need to be instantiated (e.g., load model parameters). While we investigated this aspect quantitatively with RQ2, we went into more detail and analyzed model instantiation qualitatively (e.g., what and how APIs are used), to understand their integration into traditional code.

Model In- & Output: *What is the input and output of ML models? What data types are used?* How ML models are embedded into a system also depends on the data being processed. We examine the type of input and output data to understand what data is used in ML-enabled functionality.

Pre- & Postprocessing: *What pre-and postprocessing steps are implemented for the ML models?* Since different parts of a system typically use different data formats, to actually use data as input for an ML model, and to use its output, it often needs to be processed before and after. Also, other functions that need to be performed on this data, such as safeguards [76], [77], are often considered essential. This aspect aims to understand the specifics needed for integrating an ML model.

Interaction Patterns. Our final aspect addresses ***RQ3.2*** *(What are integration patterns of ML models?).*

Model Interaction: *How do multiple models interact?* We specifically analyzed systems that contain multiple ML models with respect to the interaction of their models. As described above, we followed call graphs to comprehend the code and to determine data flows between the models. We sketched the topologies and when similar ones appeared in at least two systems defined these as a pattern. The patterns from the case study by Peng et al. [25] provided a basis.

E. Threats to Validity

Internal Validity: Personal biases of the authors, e.g., expectancy bias, may have affected the selection of subject systems and how these were analyzed quantitatively. To mitigate this threat, we based selection criteria on external sources, such as studies that identify the most popular ML libraries [41]. Similarly, we built up our analysis on previous works [39], [25] and reused existing tooling, e.g., JPlag [69], to avoid internal bias in the analysis. The classification of the 160 systems may be subject to author bias, potentially impacting how the systems are classified, and different perspectives of the labeling authors may impact the validity of the labels. To address author bias, we involved multiple authors in labeling the systems and held frequent discussions on the labels, involving a third author as mediator. To address different perspectives, we only continued with independent labeling after reaching a sufficiently high Krippendorff's Alpha. Author bias may also affect the manual analysis of the selected systems (Sec. III-D), i.e., impacting what is considered relevant. To mitigate this threat, we structured the analysis using aspects and questions, and frequently discussed our findings among three authors.

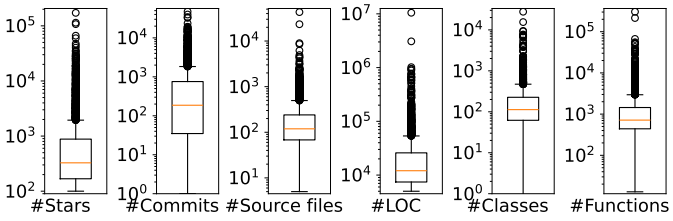

Fig. 2: General statistics of the systems in our dataset

External Validity: Our quantitative analysis is limited to Python systems and three ML libraries (PyTorch, TensorFlow, and Scikit-Learn), which threatens the generalizability of the results. However, since both Python and the selected libraries are most widely used for ML-enabled systems [63], [64], [41], our research covers the majority of relevant systems. Due to the sample size of 160 of 2,928 systems, the observed distributions of system type and relation to ML are only representative with a relatively high margin of error (up to 7.43 % at the 95 % confidence level for *Conceptually Applied*). The manually analyzed subset of systems is so small that we can only present qualitative observations. Our automated analysis relies on parsing Python code into ASTs, which failed for individual files, but the error rate (on average 0.06 % of the files) is negligible. Analyzing the number of ML-related functions is difficult due to the use of wrappers, e.g., one function loads a model and another one uses it. As a result, we only show a lower bound on the number of ML functions, and more accurate qualitative observations as part of our manual analysis. The sometimes generic names of the APIs used to load pre-trained models may lead to false positives when identifying the use of pre-trained models. To address this, we only consider API calls in the context of a model hub and are not ambiguous about the loaded object.

IV. System Characteristics (RQ1)

We now present characteristics of the systems, including their types, ML assets, and quality assurance practices.

Descriptive Statistics. We mined 2,928 ML-enabled systems that vary in scale significantly (see Fig. 2). On average, they have 263 source files (median 119), 34,000 LOC (median 12,000), 252 classes (median 113), and 1,171 functions (median 715). They are of substantial popularity, with an average of 1,487 stars (median 327) and 829 commits (median 187). Outliers are The Algorithms (181k stars), a collection of ML algorithms for educational purposes, and Azure SDK for Python with the largest codebase of 10M LOC. Figure 3 shows the used ML libraries. PyTorch is most common (78 % of systems), followed by scikit-learn and TensorFlow (52% / 38% of systems). More than half of the systems use more than one library, indicating that different types of ML are used.

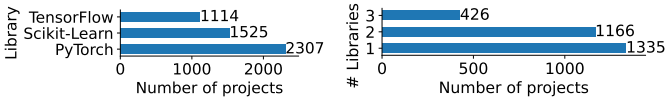

Fig. 3: ML libraries used in the ML-enabled systems

TABLE I: Types of systems and their relation to ML

	total[1]	Application	Library	Framework	Plugin
total[1]		32	99	32	2
Business	56	26	29	4	2
Conceptual	67	2	56	9	0
ML-Tool	60	8	27	27	0

[1] Total number of systems of a given type of relation to ML, independent of the other dimension. Each system can have multiple labels in each category.

A. System Classification (RQ1.1)

Upon our random sample of 160 systems we derived labels for system types and their relation to ML. Our appendix [34] has detailed label descriptions, examples, and labeling guidelines. Table I shows the results of the classifications.

System Types. We define the following four types of systems.

Applications are standalone, executable systems that provide end-user-oriented functionality through a user interface (e.g., GUI or command line). They do not require users to have programming skills. Applications solve problems and provide solutions to end-users, implementing business logic to do so.

Libraries provide functionality intended to be used in other programs via code APIs. Except for the provided functionality they contain no application logic and cannot run standalone.

Frameworks provide general application logic, but are designed to be extended by concrete functionality orchestrated by the framework. To this end, frameworks take control of the code that uses the framework, according to the principle of dependency inversion [78]. Similar to libraries, frameworks usually provide generic helper functions that are frequently used in the context of the framework.

Plugins extend applications with functionality via extension points, and only function through them. Users interact with plugins through the UI of the applications.

Our dataset consists mainly of libraries (62 %), followed by applications and frameworks (20 %), and almost no plugins.

Relation to ML. We identified three ways in which an ML-enabled system can be related to ML, i.e., how these use ML.

Business-Focused Applied refers to systems that apply ML on a real-world problem and provide a benefit to end users. They are generally focused on processing or generating data, linked with business logic to encapsulate ML.

Conceptually Applied refers to systems that demonstrate how ML technologies can be used to solve a variety of problems. They are focused on evaluating an ML model from a data scientist's perspective and have a technical focus.

ML Tools refers to tools that support building ML models, usually libraries or frameworks that implement ML functions and algorithms, such as loss functions and optimizers (e.g. TensorFlow), but also tools that support the general ML development processes, such as experiment management tools [79].

The relation of the systems to ML is almost equally distributed, with 42 % of the systems using ML conceptually, 38 % supporting building ML models, and 35 % actually using ML to provide end-user-oriented functionality.

We further investigated common combinations of project type and relation to ML. Notably, the majority of systems are

libraries that apply ML conceptually (35 %). Most applications (81 %) and all plugins are business-focused, but most business-focused systems are libraries (51 %). Frameworks are mainly ML tools that allow building ML models (84 %).

Overall, ML projects on GitHub appear to mainly demonstrate ML functionalities and provide it to other software.

B. Extent of ML-Specific Assets and Relation to Code (RQ1.2)

ML Assets. The most prevalent assets in our full dataset are model implementations in code, which are present in 2,499 systems (85 %). The average number of classes per project that inherit from "*Module*" or "*BaseEstimator*" is 85, the median is 34. These classes are either complete models or components. Nevertheless, it is safe to assume that the average project contains multiple ML models, since one model is unlikely to be composed of 34 modules. Additionally, entire ML models can be loaded from binary files, containing the weights for configuring models in code. These are, however, only contained in 339 repositories (12 %), with each of these containing on average 18 such files (median 2). Figure 4 gives an overview of implemented modules and model binary files. Since some forms of ML (e.g., PCA, clustering) requires no models to be stored, some systems contain no such assets.

Relation of ML and Non-ML Code. To determine the extent of ML-related code, we measured the amount of ML-related and non-ML-related code in our full dataset. As shown in Fig. 5, on average 42 % of the source files (median 45 %) and 34 % of the functions (median 36 %) are ML-related. So traditional code still makes up most of the systems.

Our dataset contains systems with different relationships to ML, including those that apply ML to deliver functionality (business focused), those that demonstrate possible applications (conceptual), as well as ML tools, that support the development of other systems. We investigated on our manually classified sample how this affects the share of ML-related files and functions. While the share of ML code is the highest for conceptual ML systems (on average 50 % of files and 38 % of functions), even in these systems, traditional source code, such as utility functions, data preprocessing, tests, and demonstrations, makes up most of the system. In business-focused-applied ML systems, code related to ML is the lowest (on average 31 % of files and 24 % of functions) among all types of systems investigated, since ML is used only for individual features. Similarly, ML tools provide many additional features, such as UIs, but usually focus more on ML technology. For these systems, the average share of ML code is 35 % of the files and 29 % of the functions, which is in between the other two categories.

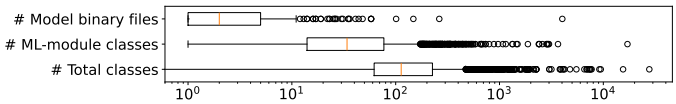

Fig. 4: Number of ML modules/ model binaries in the systems

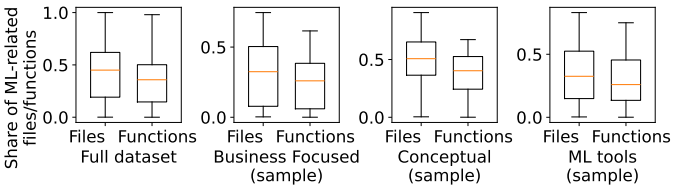

Fig. 5: Share of ML code for the full dataset and the classified sample, according to the projects' relation to ML

C. Testing (RQ1.3)

To investigate testing practices, on the full dataset, we analyzed the prevalence of unit tests and the proportion of functions directly covered by them. The overall prevalence of unit tests is very low, with only 954 of the subject systems (33 %) containing at least one unit test. We calculated the number of functions invoked in these test cases. Among the systems, containing at least one test file, the units tests directly cover a median of 6 % of all functions and 5.6 % of the ML-functions (see Fig. 6). Test coverage of ML-enabled systems is generally rather low with no differences between ML and non-ML code. While 90 % of the systems train custom ML models, 94 % evaluate their models. To this end, training is implemented in a median of 5 files, and evaluation in 9 files. Accordingly, some systems do not trust pre-trained models, but evaluate them in their application context.

Summary *RQ1*: System Characteristics

The majority of ML-enabled systems on GitHub are libraries and frameworks. End-user-oriented applications are still a minority. The systems use ML in different ways, either developing new ML technology or applying ML on concrete problems. A small, but still substantial, number of tools that support ML engineering exist. The ML-enabled systems vary in size and contain many different ML assets, but also large amounts of non-ML code. Quality assurance is neglected.

V. Reuse of ML Models and Code (RQ2)

We now discuss the extent of model reuse in the form of pre-trained models and duplicated model implementations.

A. Prevalence of Pre-Trained Models (RQ2.1)

Pre-trained models, loaded from model hubs, are used in 1,209 of the 2,928 systems (41%). The systems contain, on average, 29 API calls to the model hubs. Figure 7 shows the results.

Due to the high number of API calls per system, we investigated individual systems manually, finding that often not full models are loaded but components that are used together (e.g., encoder and decoder models in Transformers or generator and discriminator in Generative Adversarial Networks).

By analyzing the directory names of code files with API calls, we determined the location in the system and potential

Fig. 6: Test coverage for systems with ≥ 1 unit test

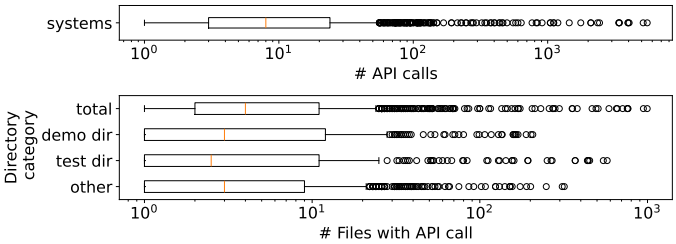

Fig. 7: Number of API calls for loading of pre-trained models per system and number of files with API call per category

purpose for which models are loaded. Out of all files that contain an API call to load pre-trained models, per system, 8.3 % are located in test directories, and 9.8 % in directories named "demo" or "experiment." The remaining 81.9 % are located in other directories containing code, such as model implementations, utility functions, or application logic.

ML models are loaded in 809 systems for neither testing nor demonstration purposes. 80 systems load pre-trained models only in demonstration directories. This can be explained by the fact, that some systems in our dataset are *ML Tools* (cf. Sec. IV-A), which do not implement ML-based functionality, but might demonstrate pre-trained models.

B. Extent of ML Asset Duplication (RQ2.2)

To determine how model implementations are reused, we examined for each system, how much of the source code implementing ML models is duplicated from other systems in the dataset. We found this type of reuse in 1,690 of the subject systems (58%). These contain an average of 18 files that are at least partially duplicated, which is more than half of the average of 34 model implementation files in the subject systems. We also observed that a few repositories provide the basis for most copies, and that many of these belong to the same organizations, notably Microsoft, OpenMMLab, and Hugging Face. Table II shows the 10 projects from which is copied the most.

Summary *RQ2*: Model Reuse

Reuse of ML models is prevalent among the subject systems. Of the analyzed systems, 1,209 use pre-trained models and 1,690 systems copy ML implementation code. The copied code originates mostly from a small set of repositories maintained by an even smaller set of organizations.

VI. Architectural Aspects (RQ3)

We answer ***RQ3*** based on insights from our in-depth analysis of 26 ML-enabled applications (cf. Sec. III-D).

TABLE II: Overview of the 10 most duplicated projects

#[1]	original project	#[1]	original project
301	microsoft/unilm	164	microsoft/LoRA
266	microsoft/LMOps	145	locuslab/convmixer
227	iscyy/yoloair	142	open-mmlab/mmdetection
193	huggingface/transformers	108	facebookresearch/fairseq
172	open-mmlab/mmsegmentation	107	Stability-AI/stablediffusion

[1] Number of projects that copy model code from the given original.

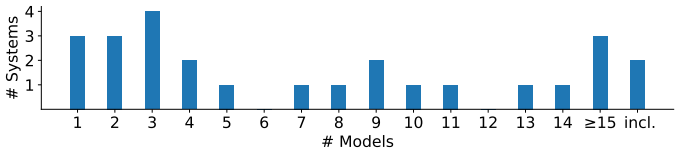

Fig. 8: Distribution of the number of models (sample)

A. Descriptive Aspects of ML Models

We start by characterizing the role of ML models in the systems, contextualizing the results of ***RQ1*** and ***RQ2***.

ML Functionalities and Tasks. Common functionalities provided by ML models include *image processing* (e.g., tiatoolbox), *chatbots* (e.g., Gentopia), *video processing* (e.g., VideoTo3dPoseAndBvh), *data analysis* (e.g., CorpusTools), and *robotic navigation* (e.g., sunnypilot). Other systems use ML to transcribe music, infer metadata, and play games. Table III shows the distribution of ML functions and tasks.

On a technical level, the underlying ML tasks in Table III are *generative AI* (e.g., h2ogpt is a chatbot), *classification* (e.g. falldetection_openpifpaf classifies videos), *decision making* (e.g., sunnypilot), *segmentation* (e.g., VideoTo3dPoseAndBvh segments videos), *dimensionality reduction* (e.g., CorpusTools), with 2 systems not fitting into established categories. For example, tournesol uses *linear regression* to visualize correlations. Four systems combine tasks, for instance, Video-ChatGPT is a *chatbot* with *video processing* capabilities that allows user to query the content of videos.

Number and Type of ML Models. The majority of the systems employ multiple ML models (see Fig. 8), with the highest number being 18 (h2ogpt). Half of the systems use more than 5 models. Three systems allow to select arbitrary models from model hubs, making the number of models practically infinite. For example, gentopia allows the user to configure a *chatbot* by selecting a model from Hugging Face.

The ML technologies found are shown in Table III. The most common type of model is a Deep Neural Network (DNN, i.e., DL) found in 21 systems. *Classical ML* (not DL) was found 5 times. Among the DNNs, we observed multiple variations, including 8 *Transformers*, 6 *Convolutional Neural Networks (CNN)*, 3 *Fully Connected Neural Networks (FCNN)*, and one *Long Short Term Memory (LSTM)* [80]. *CNNs* are used exclusively for *image processing*, whereas Transformers and LSTMs are mainly intended for natural language processing, although we found 3 Transformers that are used for *image generation* (e.g., ControlLoRA).

TABLE III: ML functions, tasks, and technologies (sample)

ML Function	#[1]	ML Task	#[1]	ML Technology	#[1]
Image processing	6	Generative AI	11	Transformer	8
Video processing	5	Classification	6	CNN	6
Chatbot	5	Decision Making	4	FCNN	5
Data analysis	4	Segmentation	4	LSTM	5
Navigation	3	Dimensionality Reduction	3	PCA	3
		Clustering	1	RL	3
				Clustering	1

[1] Number of systems (assignment is not exclusive, so the numbers do not add up to 26)

TABLE IV: Storage and origin of ML models (sample)

			Model Origin		
		Total[1]	Pre-trained	Fine-tuned	Custom
Model Storage	Total[1]		11	7	11
	Local	15	7	4	7
	Remote	15	9	4	2
	NA	3	0	0	3

[1] Number of systems with a given origin or storage, independent of the other dimension; a system can have multiple models with different origin or storage.

ML can be further divided into *Supervised Learning (SL)*, *Unsupervised Learning (UL)*, and *Reinforcement Learning (RL)* [81]. UL was found only in the form of classical ML, such as *Principal Component Analysis (PCA)* and *Clustering*. Two systems use supervised classical ML. The DNNs are mostly used for SL, but 3 RL systems were also found. In contrast to DL, classical ML is mostly used for auxiliary functions, e.g., CorpusTools uses PCA for data visualization.

Model Origins. 11 systems use pre-trained models directly out-of-the-box, while 7 systems fine-tune them, and 11 systems use completely custom-implemented models. DNNs are usually pre-trained or fine-tuned, whereas classical ML models are always custom-trained. The likely reason is that the more complex a model becomes, the more training data and effort it requires, so pre-trained models relieve developers of end-user-oriented systems of this burden [27]. We found 6 systems that use custom-trained DL models, 3 of which use RL, for which few pre-trained models exist. Other custom built ML models are used for specialized use cases, such as DreamArtist-stable-diffusion, which uses custom models for prompt optimization.

Storing & Loading of Models. We observed local and remote storage 15 times each, with 7 systems supporting both. For example, Vlog uses pre-trained models from different sources, some stored locally, some downloaded from Hugging Face. Since UL does not need models to be stored, 3 systems have no storage. We visualize the relationship between storage and origin of the models as a matrix in Table IV. Custom models are always stored locally, pre-trained models usually remotely, with some exceptions, such as stable-diffusion-webui-depthmap-script or VideoTo3dPoseAndBvh.

B. Model Embedding (RQ3.1)

ML models are embedded in traditional code by using code-level APIs. The main interaction with models is via their inputs and outputs, which often require pre- or postprocessing. However, we observed that developers often resort to ad hoc solutions to embed ML models. We report on observed inputs and outputs, and pre- and postprocessing practices.

Model In- & Output. We found a wide range of data types used as inputs and outputs to ML models (see Table V). Inputs include text (e.g., paperless-ng), images (e.g., imagepy), and video (e.g., VideoTo3dPoseAndBvh). Less frequent are audio (sheetsage) or an environment state for RL agents, in the form of structured numeric data (deep_learning_and_the_game_of_go or sarl_star). The outputs are more diverse, commonly text (e.g., sheetsage), images (e.g., clip-glass), labels (e.g., paperless-ng, which classifies text documents), actions (e.g., sunnypilot, or other robotics systems), bounding boxes (e.g., VIAME), and video, (e.g., VLog). While we mostly observed standard data formats, special application scenarios sometimes require less common formats, e.g., VideoTo3dPoseAndBvh extracts animations from video and saves in the BVH format. Structured numeric data is outputted by the model of Sunnypilot containing inferred information about the environment, like the lane lines or the distance to the lead vehicle. The most common combination of input and output is text to text (4 systems, all chatbots). Other common combinations are text to image or image to text. clip-glass is an example of both, providing image captioning and image generation from text.

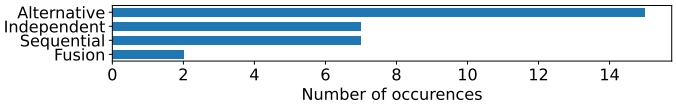

Fig. 9: Distribution of model interaction patterns (sample)

Pre- & Postprocessing. Almost all systems contain pre- and post processing steps, ranging from generic operations (e.g., text tokenization) to more complex content manipulations, often tailored to the needs of a specific system. We clearly identified preprocessing code in 15 systems; 4 systems contained no pre-processing; while the remaining 7 were inconclusive. Common preprocessing steps include normalization, feature extraction, and vectorization. Some processing functions such as image normalizations were scattered throughout the system.

Postprocessing was found in 8 systems, 9 contain none, 9 were inconclusive. Again, the majority are simple data conversions, such as projecting tokens back to text. More complex tasks were also observed, e.g., validating actions chosen by an RL agent (deep_learning_and_the_game_of_go).

Particularly interesting is tiatoolbox, which provides users with extensive configuration options, including pre- and post-processing methods. While it provides default functions, it also supports the integration of user-written functions added to the model as callbacks. DreamArtist-stable-diffusion, implements some security functionalities that ensure input integrity.

C. Interaction Patterns (RQ3.2)

In the 23 systems with multiple models (see Sec. IV), we observed various types of interaction between these models. Triangulating with related work [25], [59], we extracted 4 interaction patterns that describe the observed topologies. Figure 9 shows the frequency of each pattern in the sample.

Alternative use of models is the most common pattern (15 systems), where multiple models are offered for the same task and chosen either automatically or by the user. For example, sarl_star provides multiple RL agents to control a robot.

TABLE V: Data types as input and output to models (sample)

	Text	Image	Video	Labels	Actions	Numeric Data	Bounding Boxes	Audio
Input	10	10	5	-	-	2	-	1
Output	8	5	2	4	4	-	2	-

Independent models for unrelated functionalities are found in 7 systems. Larger systems in particular provide the user with multiple functions, some of which require ML. For example, VIAME offers *image processing* tools (e.g., for *segmentation* or *classification*) to be used independently by the user.

Sequential integration of ML models was observed in 7 systems. The output of one model is used as input to another model, possibly with additional processing steps in between. This pattern is used for more complex tasks, such as perception systems; e.g., home-robot uses a model to processes sensor input and feeds the results (i.e., detected obstacles) to an RL agent for navigation. Sometimes, additional data is added to the subsequent models, e.g, Video-ChatGPT uses a Transformer model to summarize videos, which is the input for a Large Language Model (LLM) together with text prompts.

Joining integration of ML models is the most complex pattern and was found twice in different variants. Results of two or more models are combined, either by another model (which also includes the sequential pattern) or code logic. We observed one variant in Vlog, which has a complex pipeline of 6 models to transform videos into text descriptions. It includes a model transcribing audio and one generating captions for each frame. Both results are fed into an LLM, which assembles a final text document. The other variant was observed in eynollah, where two segmentation models process an image, with each model being used for different parts of the image. The segmented image is further processed by other models.

Multiple of these patterns are implemented in 8 systems. For instance, VIAME offers multiple *independent* ML functionalities, but has *alternative* models for some of them. In Vlog, data processing starts with a feature extractor model, followed by a segmentor (*sequential*) to divide a video into chunks. The chunks are then processed by two models, one for subtitle generation and the other for segmentation and classification. In parallel, the audio track is translated into text by a third ML model. The outputs of the three models are then assembled by an LLM (*joining*) into a text description of the video.

Summary *RQ3*: Architectural Aspects

Most systems use supervised DL with pre-trained models, followed by deep reinforcement learning and classical unsupervised learning. While most ML models process text or images, other types of input are also present. Using multiple models is very common. Although most models are either used as alternatives or independently, some systems apply multiple models sequentially. Pre-and postprocessing steps are scattered throughout the codebase and there are no common practices for integrating ML models.

VII. Related Work

Prior research has characterized ML-enabled systems and examined challenges related to the integration of ML and relevant aspects. We complement with the first investigation of model reuse and architectural aspects on a large dataset.

Similar to how we study **system characteristics**, Gonzales et al. [64] analyze ML projects, compare them to non-ML projects, and identify unique properties, such as programming languages, library dependencies, and developer collaboration. We confirm some of their findings, for instance, that ML tools are still more popular than applied ML projects. Our study, however, has a much wider scope, including a qualitative analysis that we relate to the quantitative analysis.

Jiang et al. [82] study **model reuse** on a dataset of pre-trained models. Their dataset is limited, considering only two major model hubs, whereas we consider seven. We also investigate reuse of ML assets through code duplication.

Gorton et al. [15] suggest a conceptual **ML architecture**, which we did not observe in the analyzed systems. Peng et al. [25] examine the architecture of an autonomous driving system and identify patterns of interaction between ML models and source code. We complement their findings with a broader perspective, covering 26 ML-enabled systems. Similar to us, Nahar et al. [40] combine repository mining with a manual analysis of a subset. They find that only a fraction of systems using multiple ML models contain interacting models. However, their analysis is at a more abstract level than ours, focusing on processes and collaboration rather than implementation details such as the type of ML model used. They also focus on qualitative methods, while we combine these with quantitative analysis.

Houerbi et al. [83] studied **CI pipelines** in open source ML projects and identified significant issues. In fact, in our dataset, 1,091 of 2,928 repositories (37%) use continuous integration (883 use GitHub actions, 192 Travis, and 16 Jenkins), confirming their observations about adoption. Openja et al. [45] investigate **testing practices** by manually analyzing test files from 11 open-source ML projects, highlighting used test types and tested ML-related functionalities.

While we took a static view on ML-enabled systems, others analyze their **evolution**, focusing on how ML assets evolve in relation to source code [84], and the types of changes contributed by forks of ML repositories [21]. Simmons et al. [85] analyze issues in open-source ML projects, finding that ML-related issues take longer to resolve than non-ML issues.

Finally, Munappy et al. [86] investigate challenges in **data management** for deep learning models, identifying 20 challenges and possible solutions. Biswas et al. [66] characterize data-science pipelines, finding that in practice they differ from each other and are often more complicated than the theoretical models. This is, however, beyond the scope of our study.

VIII. Discussion

We now discuss our findings and formulate actionable recommendations for researchers and practitioners.

ML Adoption in Open-Source Software. Despite growing attention, the adoption of ML in open-source systems is lower than expected. Most systems are prototypes (e.g., ConsistentTeacher, a research prototype to improve object detection) and proofs of concept, demonstrating new ML technologies (e.g., disrupting-deepfakes, which demonstrates how images can be protected from manipulation). Applications providing end-user-oriented functionality (e.g., paperless-ng, a tool for organizing scanned text documents by inferring

metadata) are a minority and often appear prototypical as well (e.g., home-robot refers to itself as a research tool). This aligns with others' findings [40], which suggest that open-source ML applications often resemble startup-style projects where ML is not a central component.

While the analyzed applications offer diverse capabilities based on ML, these are often based on ad hoc solutions to integrate ML models (e.g., falldetection_openpifpaf uses 19 if-statements to load alternative models). This indicates that developers are still exploring the possibilities of ML (cf. ***RQ1.1***), and highlights the need for design methods that consider the specifics of ML and more structured techniques to integrate ML models into software systems (cf. ***RQ3.1***).

Recommendation

Effective ML adoption requires a holistic perspective on building ML-enabled systems. Further studies are needed to understand ML adoption barriers (e.g., lack of use cases, engineering practices, or resources), and develop methods and tools for adoption.

ML Asset Reuse. ML assets are extensively reused across software systems, often employing bad reuse practices, such as code cloning. We identified reuse as an essential practice, as 65 % of systems reuse ML assets. This reuse can be easily motivated, e.g., due to lack of data science expertise among software engineers [87]. However, the observed reuse is rarely systematic, i.e., cloning model code, which poses maintainability issues—as observed in code clones in general [88], [89]. Although pre-trained models should provide a manageable form of reuse, 75 % of the systems using pre-trained models contain also code clones. While we confirm the importance of pre-trained models in practice, other studies also highlight issues, such as insufficient security measures, missing attributes or artifacts, and discrepancies in model performance [55], [46]. We show that code duplication is not predominant in a single ML technology, but prone to all areas, i.e., transformers, stable diffusion, and others (cf. Table II). Our in-depth analysis of a sample of 26 ML-enabled applications, revealed no other patterns of model reuse. A possible reason for cloning model implementations is that developers modify the model implementations slightly, which is supported by the fact, that many of the identified code duplicates are not complete matches. However, this may also result from duplicates using outdated model versions.

Recommendation

Increase developers' awareness of proper reuse mechanisms, particularly for ML assets, and promote their adoption. This must be accompanied by academic studies to identify the reasons for the observed bad practices and the development of effective, easy-to-use reuse mechanisms.

Topologies of ML Models. *Many ML-enabled systems integrate multiple models. However, the actual embedding of ML models into the systems and the integration of multiple models is realized in an ad hoc manner. Best practices are missing.* In line with other studies [25], [40], the analyzed systems often employ multiple ML models that interact in various ways through complex code logic. For example, Nahar et al. [40] show that 60 % of th systems they investigated have multiple models and 23 % contain interacting models. In our sample, even 88 % of the systems contain multiple models, and 69 % involve some form of interaction. Still, we did not observe any systematic integration strategy, but ad hoc implementations. Especially pre- and post processing of data lacks clear structure (e.g., tiatoolbox allows for arbitrary pre-and postprocessing methods, other systems, e.g., Gentopia contain none). Other observations include heavy use of wrappers for loading ML models, and extensive IF-statements for loading alternative models (falldetection_openpifpaf contains an IF with 19 alternatives). This lack of methods, missing architectures or templates, may be a core reason why advancements in ML do not carry over to building systems (in fact, most projects stall or fail in the development phase [9]). Researchers can assist by providing guidelines on best practices and architectures for developing complex topologies of ML models. Additionally, tools and libraries, such as model hubs, should be improved to provide adequate support for developing ML-enabled systems, particularly considering multiple ML models. Current tools, such as experiment management tools, are not only model-centric, but lack support for multiple models [79], [16].

Recommendation

Complex ML model topologies are part of ML-enabled systems and need to be considered in future research. This requires design patterns and tools that can manage multiple models, lifting the current model-centric to a system-centric perspective in the development of ML-enabled systems.

IX. Conclusion

We presented a large-scale study of ML model integration in ML-enabled software systems. We quantitatively analyzed 2,928 systems in terms of their characteristics, comparing ML and non-ML assets and reuse practices. We manually classified a random sample of 160 systems in terms of their relationship to ML and system type, and analyzed the 26 applications of this sample in depth regarding their provided functionality, model reuse practices, and architectural aspects.

Among others, we find that ML is used for a variety of functionalities, but mostly in conceptual prototypes. We discovered extensive, but unstructured reuse of ML models. Although we observed a lack of structured approaches to the architecture of ML-enabled systems, we identified four patterns of interaction between ML models. We contribute recommendations for researchers and tool builders.

Acknowledgment

We thank Kevin Hermann for providing feedback and Constanze Ohrem for the help with labeling the dataset. This work was partially funded by the German Federal Ministry for Education and Research (BMBF) under the project PrivacyE2E.

REFERENCES

[1] A. Chernikova, A. Oprea, C. Nita-Rotaru, and B. Kim, "Are Self-Driving Cars Secure? Evasion attacks against deep neural networks for steering angle prediction," in *Security and Privacy Workshops (SPW)*, 2019.

[2] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, "Deep Learning for Object Detection and Scene Perception in Self-Driving Cars: Survey, Challenges, and Open Issues," *Array*, vol. 10, p. 100057, 2021.

[3] M. T. Chowdhury and J. Cleland-Huang, "Engineering Challenges for AI-Supported Computer Vision in Small Uncrewed Aerial Systems," in *International Conference on AI Engineering – Software Engineering for AI (CAIN)*, 2023.

[4] A. Vellido, "The Importance of Interpretability and Visualization in Machine Learning for Applications in Medicine and Health Care," *Neural Computing and Applications*, vol. 32, no. 24, pp. 18069–18083, 2020.

[5] J. W. Goodell, S. Kumar, W. M. Lim, and D. Pattnaik, "Artificial Intelligence and Machine Learning in Finance: Identifying Foundations, Themes, and Research Clusters from Bibliometric Analysis," *Journal of Behavioral and Experimental Finance*, vol. 32, p. 100577, 2021.

[6] J. Ryseff, B. F. D. Bruhl, and S. J. Newberry, "The Root Causes of Failure for Artificial Intelligence Projects and How They Can Succeed: Avoiding the Anti-Patterns of AI." https://www.rand.org/pubs/research_reports/RRA2680-1.html#document-details, 2024. [Accessed 2024-09-29].

[7] J. Kahn, "Want Your Company's A.I. Project to Succeed? Don't Hand it to the Data Scientists, Says this CEO." https://fortune.com/2022/07/26/a-i-success-business-sense-aible-sengupta/, 2022. [Accessed 2024-09-29].

[8] insideAI News, "Survey: 96% of Enterprises Encounter Training Data Quality and Labeling Challenges in Machine Learning Projects." https://insideainews.com/2019/05/26/survey-96-of-enterprises-encounter-training-data-quality-and-labeling-challenges-in-machine-learning-projects/, 2019. [Accessed 2024-08-02].

[9] B. T. O'Neill, "Failure Rates for Analytics, AI, and Big Data Projects = 85% – Yikes!." https://designingforanalytics.com/resources/failure-rates-for-analytics-bi-iot-and-big-data-projects-85-yikes/, 2019. [Accessed 2024-08-02].

[10] N. Hotz, "Why Big Data Science & Data Analytics Projects Fail." https://www.datascience-pm.com/project-failures/, 2024. [Accessed 2024-08-02].

[11] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software Engineering for Machine Learning: A Case Study," in *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019.

[12] D. K. Becker, "Predicting Outcomes for Big Data Projects: Big Data Project Dynamics (BDPD): Research in Progress," in *International Conference on Big Data*, 2017.

[13] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software Engineering Challenges of Deep Learning," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2018.

[14] J. Bosch, H. H. Olsson, and I. Crnkovic, "Engineering AI Systems: A Research Agenda," *Artificial Intelligence Paradigms for Smart Cyber-Physical Systems*, pp. 1–19, 2021.

[15] I. Gorton, F. Khomh, V. Lenarduzzi, C. Menghi, and D. Roman, "Software Architectures for AI Systems: State of Practice and Challenges," in *Software Architecture: Research Roadmaps from the Community*, pp. 25–39, 2023.

[16] S. Idowu, O. Osman, D. Strüber, and T. Berger, "On the Effectiveness of Machine Learning Experiment Management Tools," in *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2022.

[17] S. Idowu, D. Strüber, and T. Berger, "Asset Management in Machine Learning: State-of-Research and State-of-Practice," *ACM Computing Survey*, vol. 55, no. 7, pp. 1–35, 2022.

[18] Z. Zhao, Y. Chen, A. A. Bangash, B. Adams, and A. E. Hassan, "An Empirical Study of Challenges in Machine Learning Asset Management," *Empirical Software Engineering*, vol. 29, no. 4, p. 98, 2024.

[19] I. Alves, L. A. F. Leite, P. Meirelles, F. Kon, and C. S. R. Aguiar, "Practices for managing machine learning products: A multivocal literature review," *IEEE Transactions on Engineering Management*, vol. 71, pp. 7425–7455, 2024.

[20] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden Technical Debt in Machine Learning Systems," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[21] A. Bhatia, E. E. Eghan, M. Grichi, W. G. Cavanagh, Z. Ming, B. Adams, *et al.*, "Towards a Change Taxonomy for Machine Learning Systems," *arXiv*, vol. abs/2203.11365, 2022.

[22] H. B. Braiek and F. Khomh, "On Testing Machine Learning Programs," *Journal of Systems and Software (JSS)*, vol. 164, p. 110542, 2020.

[23] R. Nazir, A. Bucaioni, and P. Pelliccione, "Architecting ML-Enabled Systems: Challenges, Best Practices, and Design Decisions," *Journal of Systems and Software (JSS)*, vol. 207, p. 111860, 2024.

[24] N. Nahar, H. Zhang, G. Lewis, S. Zhou, and C. Kästner, "A meta-summary of challenges in building products with ml components–collecting experiences from 4758+ practitioners," in *2nd International Conference on AI Engineering–Software Engineering for AI (CAIN)*, 2023.

[25] Z. Peng, J. Yang, T.-H. P. Chen, and L. Ma, "A First Look at the Integration of Machine Learning Models in Complex Autonomous Driving Systems: A Case Study on Apollo," in *European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020.

[26] D. Hendrycks, K. Lee, and M. Mazeika, "Using pre-training can improve model robustness and uncertainty," in *International Conference on Machine Learning*, 2019.

[27] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why Does Unsupervised Pre-Training Help Deep Learning?," in *International Conference on Artificial Intelligence and Statistics*, 2010.

[28] S. Idowu, D. Strüber, and T. Berger, "EMMM: A Unified Meta-Model for Tracking Machine Learning Experiments," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2022.

[29] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, and S. Wagner, "Software Engineering for AI-based Systems: A Survey," *Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–59, 2022.

[30] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained Models for Natural Language Processing: A Survey," *arXiv*, vol. abs/2003.08271, 2020.

[31] S. Long, F. Cao, S. C. Han, and H. Yang, "Vision-and-Language Pretrained Models: A Survey," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2022.

[32] W. Frakes and K. Kang, "Software Reuse Research: Status and Future," *Transactions on Software Engineering (TSE)*, vol. 31, pp. 529–536, 2005.

[33] I. J. Mojica, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. E. Hassan, "A Large-Scale Empirical Study on Software Reuse in Mobile Apps," *IEEE Software*, vol. 31, no. 2, pp. 78–86, 2013.

[34] Y. Sens, H. Knopp, S. Peldzus, and T. Berger, "Replication Package." https://doi.org/10.5281/zenodo.14169777, 2025.

[35] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech, "Generative AI," *Business & Information Systems Engineering*, vol. 66, no. 1, pp. 111–126, 2024.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[37] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional Neural Networks: An Overview and Application in Radiology," *Insights into Imaging*, vol. 9, pp. 611–629, 2018.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is All You Need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[39] S. Idowu, Y. Sens, T. Berger, J. Krüger, and M. Vierhauser, "A Large-Scale Study of ML-Related Python Projects," in *Symposium On Applied Computing (SAC)*, 2024.

[40] N. Nahar, H. Zhang, G. Lewis, S. Zhou, and C. Kästner, "The Product Beyond the Model – An Empirical Study of Repositories of Open-Source ML Products," in *International Conference on Software Engineering (ICSE)*, 2024.

[41] I. Stančin and A. Jović, "An Overview and Comparison of Free Python Libraries for Data Mining and Big Data Analysis," in *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2019.

[42] PyTorch, "Open Source Machine Learning Framework." https://pytorch.org, 2023. [Accessed 2023-07-01].

[43] TensorFlow, "End-to-End Machine Learning Platform." https://www.tensorflow.org, 2023. [Accessed 2023-07-01].

[44] scikit-learn, "Machine Learning in Python." https://scikit-learn.org, 2023. [Accessed 2023-07-01].

[45] M. Openja, F. Khomh, A. Foundjem, Z. M. Jiang, M. Abidi, and A. E. Hassan, "An Empirical Study of Testing Machine Learning in the Wild," *Transactions on Software Engineering and Methodology (TOSEM)*, vol. 34, pp. 1–63, 2024.

[46] W. Jiang, N. Synovic, R. Sethi, A. Indarapu, M. Hyatt, T. R. Schorlemmer, G. K. Thiruvathukal, and J. C. Davis, "An Empirical Study of Artifacts and Security Risks in the Pre-trained Model Supply Chain," in *Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2022.

[47] PyTorch Hub, "Pretrained-Model Repository Designed for Research Exploration." https://pytorch.org/hub/, 2024. [Accessed 2024-03-21].

[48] Hugging Face, "Platform Where the Machine Learning Community Collaborates on Models." https://huggingface.co/, 2024. [Accessed 2024-03-21].

[49] Model Zoo, "Open Source Deep Learning Code and Pretrained Models." https://modelzoo.co/, 2024. [Accessed 2024-03-21].

[50] ONNX Model Zoo, "Open Format Built to Represent Machine Learning Models." https://onnx.ai/models/, 2024. [Accessed 2024-03-21].

[51] Modelhub, "Repository of Self-Contained Deep Learning Models Pre-trained for a Wide Variety of Applications." http://modelhub.ai/, 2024. [Accessed 2024-03-21].

[52] NVIDIA NGC, "Portal of Enterprise Services, Software, Management Tools, and Support for End-to-End AI and Digital Twin Workflows." https://www.nvidia.com/en-us/gpu-cloud/, 2024. [Accessed 2024-03-21].

[53] MATLAB Model Hub, "MATLAB Deep Learning Model Hub." https://de.mathworks.com/matlabcentral/fileexchange/103650-matlab-deep-learning-model-hub, 2024. [Accessed 2024-03-21].

[54] Y. Li, Z. Zhang, B. Liu, Z. Yang, and Y. Liu, "ModelDiff: testing-based DNN similarity comparison for model reuse detection," in *International Symposium on Software Testing and Analysis (ISSTA)*, 2021.

[55] W. Jiang, N. Synovic, M. Hyatt, T. R. Schorlemmer, R. Sethi, Y. Lu, G. K. Thiruvathukal, and J. C. Davis, "An Empirical Study of Pre-Trained Model Reuse in the Hugging Face Deep Learning Model Registry," in *International Conference on Software Engineering (ICSE)*, 2023.

[56] J. L. Ribeiro, M. Figueredo, A. A. Jr., N. Cacho, and F. Lopes, "A Microservice Based Architecture Topology for Machine Learning Deployment," in *International Smart Cities Conference (ISC2)*, 2019.

[57] H. Yokoyama, "Machine Learning System Architectural Pattern for Improving Operational Stability," in *International Conference on Software Architecture Companion*, 2019.

[58] I. Ozkaya, "What Is Really Different in Engineering AI-Enabled Systems?," *IEEE Software*, vol. 37, no. 4, pp. 3–6, 2020.

[59] S. Peldszus, H. Knopp, Y. Sens, and T. Berger, "Towards ML-Integration and Training Patterns for AI-Enabled Systems," in *International Conference on Bridging the Gap Between AI and Reality (AISoLA)*, 2023.

[60] S. Apel, C. Kästner, and E. Kang, "Feature Interactions on Steroids: On the Composition of ML Models," *IEEE Software*, vol. 39, no. 3, pp. 120–124, 2022.

[61] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The Promises and Perils of Mining GitHub," in *International Conference on Mining Software Repositories (MSR)*, 2014.

[62] O. Dabic, E. Aghajani, and G. Bavota, "Sampling Projects in GitHub for MSR Studies," in *International Conference on Mining Software Repositories (MSR)*, 2021.

[63] S. Raschka, J. Patterson, and C. Nolet, "Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence," *Information*, vol. 11, no. 4, p. 193, 2020.

[64] D. Gonzalez, T. Zimmermann, and N. Nagappan, "The State of the ML-Universe: 10 Years of Artificial Intelligence & Machine Learning Software Development on GitHub," in *International Conference on Mining Software Repositories (MSR)*, 2020.

[65] R. Aghili, H. Li, and F. Khomh, "Studying the Characteristics of AIOps Projects on GitHub," *arXiv*, vol. abs/2212.13245, 2022.

[66] S. Biswas, M. Wardat, and H. Rajan, "The Art and Practice of Data Science Pipelines: A Comprehensive Study of Data Science Pipelines In Theory, In-The-Small, and In-The-Large," *arXiv*, vol. abs/2112.01590, 2021.

[67] H. Borges and M. T. Valente, "What's in a Github Star? Understanding Repository Starring Practices in a Social Coding Platform," *Journal of Systems and Software (JSS)*, vol. 146, pp. 112–129, 2018.

[68] M. Vidoni, "A Systematic Process for Mining Software Repositories: Results from a Systematic Literature Review," *Information and Software Technology*, vol. 144, p. 106791, 2022.

[69] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding Plagiarisms among a Set of Programs with JPlag," *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016–1038, 2002.

[70] M. Zakeri-Nasrabadi, S. Parsa, M. Ramezani, C. Roy, and M. Ekhtiarzadeh, "A Systematic Literature Review on Source Code Similarity Measurement and Clone Detection: Techniques, Applications, and Challenges," *Journal of Systems and Software (JSS)*, vol. 204, p. 111796, 2023.

[71] M. Novak, M. Joy, and D. Kermek, "Source-code Similarity Detection and Detection Tools Used in Academia: A Systematic Review," *Transactions Computing Education*, vol. 19, no. 3, pp. 1–37, 2019.

[72] C. Ragkhitwetsagul, J. Krinke, and D. Clark, "A Comparison of Code Similarity Analysers," *Empirical Software Engineering (EMSE)*, vol. 23, pp. 2464–2519, 2018.

[73] D. E. Rzig, F. Hassan, C. Bansal, and N. Nagappan, "Characterizing the Usage of CI Tools in ML Projects," in *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2022.

[74] N. McDonald, S. Schoenebeck, and A. Forte, "Reliability and Inter-rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice," *Human Computer Interaction*, vol. 3, pp. 1–23, 2019.

[75] K. Krippendorff, *Content Analysis: An Introduction to Its Methodology*. SAGE Publications, 4th ed., 2018.

[76] A. Poth, B. Meyer, P. Schlicht, and A. Riel, "Quality Assurance for Machine Learning: An Approach to Function and System Safeguarding," in *International Conference on Software Quality, Reliability and Security (QRS)*, 2020.

[77] H. Abdelkader, M. Abdelrazek, S. Barnett, J. Schneider, P. Rani, and R. Vasa, "ML-On-Rails: Safeguarding Machine Learning Models in Software Systems – A Case Study," in *International Conference on AI Engineering - Software Engineering for AI (CAIN)*, 2024.

[78] D. Riehle, *Framework Design: A Role Modeling Approach*. PhD thesis, ETH Zurich, 2000.

[79] S. Idowu, D. Strüber, and T. Berger, "Asset Management in Machine Learning: A Survey," in *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021.

[80] H. Hussain, P. Tamizharasan, and C. Rahul, "Design Possibilities and Challenges of DNN Models: A Review from the Perspective of End Devices," *Artificial Intelligence Review*, vol. 55, pp. 1–59, 2022.

[81] T. O. Ayodele, "Types of Machine Learning Algorithms," in *New advances in machine learning* (Y. Zhang, ed.), ch. 3, pp. 19–48, InTechOpen, 2010.

[82] W. Jiang, J. Yasmin, J. Jones, N. Synovic, J. Kuo, N. Bielanski, Y. Tian, G. K. Thiruvathukal, and J. C. Davis, "Peatmoss: A dataset and initial analysis of pre-trained models in open-source software," in *International Conference on Mining Software Repositories (MSR)*, 2024.

[83] A. Houerbi, C. Siala, A. Tucker, D. E. Rzig, and F. Hassan, "Empirical Analysis on CI/CD Pipeline Evolution in Machine Learning Projects," *arXiv*, vol. abs/2403.12199, 2024.

[84] A. Barrak, E. E. Eghan, and B. Adams, "On the Co-Evolution of ML Pipelines and Source Code: Empirical Study of DVC Projects," in *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021.

[85] A. J. Simmons, S. Barnett, J. Rivera-Villicana, A. Bajaj, and R. Vasa, "A Large-Scale Comparative Analysis of Coding Standard Conformance in Open-Source Data Science Projects," in *International Symposium on Empirical Software Engineering and Measurement*, 2020.

[86] A. R. Munappy, J. Bosch, H. H. Olsson, A. Arpteg, and B. Brinne, "Data Management for Production Quality Deep Learning Models: Challenges and Solutions," *Journal of Systems and Software*, vol. 191, p. 111359, 2022.

[87] A. Hopkins and S. Booth, "Machine Learning Practices Outside Big Tech: How Resource Constraints Challenge Responsible Development," in *Conference on AI, Ethics, and Society*, AIES '21, 2021.

[88] C. J. Kapser and M. W. Godfrey, ""Cloning Considered Harmful" Considered Harmful: Patterns of Cloning in Software," *Empirical Software Engineering*, vol. 13, pp. 645–692, 2008.

[89] E. Jürgens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do Code Clones Matter?," in *International Conference on Software Engineering (ICSE)*, 2009.