The Product Beyond the Model – An Empirical Study of Repositories of Open-Source ML Products

Nadia Nahar*†, Haoran Zhang†, Grace Lewis‡, Shurui Zhou§, Christian Kästner†
†Carnegie Mellon University, ‡Carnegie Mellon Software Engineering Institute, §University of Toronto
*nadian@andrew.cmu.edu

***Abstract*—Machine learning (ML) components are increasingly incorporated into software products for end-users, but developers face challenges in transitioning from ML prototypes to products. Academics have limited access to the source of commercial ML products, hindering research progress to address these challenges. In this study, first and foremost, we contribute a dataset of 262 open-source ML products for end users (not just models), identified among more than half a million ML-related projects on GitHub. Then, we qualitatively and quantitatively analyze 30 open-source ML products to answer six broad research questions about development practices and system architecture. We find that the majority of the ML products in our sample represent more startup-style development than reported in past interview studies. We report 21 findings, including limited involvement of data scientists in many open-source ML products, unusually low modularity between ML and non-ML code, diverse architectural choices on incorporating models into products, and limited prevalence of industry best practices such as model testing, pipeline automation, and monitoring. Additionally, we discuss seven implications of this study on research, development, and education, including the need for tools to assist teams without data scientists, education opportunities, and open-source-specific research for privacy-preserving telemetry.**

***Index Terms*—Open source dataset, machine learning products, mining software repositories, software engineering for machine learning**

I. Introduction

With the increasing popularity of machine learning (ML), more and more software products are incorporating ML components to enable various capabilities, such as face recognition in photo-sharing apps. However, incorporating ML into products is not just about developing the model; there is a significant amount of effort to integrate the model into the system, while considering aspects such as system architecture, requirements, user experience (UX) design, safety and security, system testing, and operations [1]–[3]. Developers find it challenging to convert ML prototypes into *software products with ML components (ML products)* [4]–[7]. Unfortunately, researchers rarely have access to the source code of ML products and hence face difficulty in (a) studying challenges in-depth and (b) designing and evaluating interventions (e.g., tools and practices). Currently, academic researchers rely primarily on an outside view gathered through interviews or surveys with industry practitioners [1], [5]–[9]. Some researchers perform research within a company and gain rich access [10]–[14], but are limited to a single context and can rarely share details. This inability to access and study *ML products* – not just any ML projects – poses a significant impediment to advancing research in this field, leading to a wealth of academic literature identifying challenges through professionals' testimonies, but a dearth of research offering scientifically-evaluated solutions or interventions at the intersection of software engineering (SE) and ML.

TABLE I: Sample ML Products for Analysis, from the Curated Dataset of 262 ML Products: Mobile (P1-P10), Desktop (P11-P20), and Web Applications (P21-P30)

ID	Name	Descrption	Star	Cont.	U/D*
P1	Text Fairy	OCR scanner app	751	5	10M+
P2	Seek by iNaturalist	App to identify plants and animals	92	8	1M+
P3	Pocket Code	App for learning programming	92	8	7M+
P4	ESP32 AI Camera	ESP32-CAM processing AI tasks	82	1	1K+
P5	NotionAI MyMind	App to store and search for items	182	2	1K+
P6	Organic Maps	Offline map app	4023	226	500K+
P7	VertiKin	E-commerce app	74	5	N/A
P8	FlorisBoard	Android keyboard	3503	76	N/A
P9	NewsBlur	Personal news reader	6123	83	50K+
P10	TfLite MNIST	Handwritten digits classification	214	1	N/A
P11	AWIPS	Advanced weather processing system	129	6	97K/mo
P12	Audiveris	Optical musical recognition app	932	16	8.7K/mo
P13	Datashare	Doc. analysis software for journalists	438	15	1.2K/mo
P14	Algoloop	Algorithmic trading application	67	160	N/A
P15	Subtitle Edit	Editor for video subtitles	4407	86	293K/mo
P16	DeepFaceLab	Software for creating deepfakes	35566	19	N/A
P17	Faceswap	Software for creating deepfakes	42623	80	297K/mo
P18	HO	Helper for Hattrick football manager	138	12	14M+
P19	BigBlueButton	Web conferencing system	7710	181	88K/mo
P20	PoseOSC	Realtime human pose estimation	63	2	N/A
P21	OpenBB Terminal	Investment research software	17481	136	94K/mo
P22	Coffee Grind Size	Coffee particle analyzer	402	1	N/A
P23	Celestial Detection	Classifier of celestial bodies	69	20	N/A
P24	Electricity Maps	Greenhouse gas intensity visualizer	2566	268	3M+
P25	Galaxy	Data intensive science for everyone	1021	255	187K/mo
P26	GridCal	Power systems planning software	293	14	N/A
P27	Honkling	Keyword spotting system	63	5	1.2K/mo
P28	Jitsi Meet	App for video conferencing	18813	374	10M+
P29	Code.org	Professional learning program for CS	712	132	82M+
P30	Basketball Analys.	Analyze basketball shooting pose	781	4	N/A

*Users/Downloads: There is no reliable way to calculate the number of users; we report them using multiple ways if available, such as downloads in google play-store, self-reported on website, or website traffic tracker (similarweb.com) to count average monthly users (in 'value/mo' format)

Historically, the field of SE has greatly benefited from open-source software in terms of research, practice, and education. Research on mining open-source software repositories has allowed us to create vast datasets and study aspects that would otherwise be challenging to investigate [15], [16], such as effectiveness of continuous integration [17] and pull-request-based development [18]. Open source allows researchers to test interventions on software artifacts, which has been foundational for many research areas such as program repair and software testing [19], [20]. Many innovations developed and

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00006

evaluated on open source are later adopted in industry (e.g., Facebook's adoption of program repair [21], and Google's adoption of mutation testing [22]). Beyond research and development, access to open source on a larger scale has revolutionized education, offering students and professionals the opportunity to study and illustrate practices in open-source repositories [23]–[25]. In the same manner, access to open-source *ML products* could open opportunities for research, education, and technology transfer. To this end, we identify a corpus of such ML products.

Many past studies have tried to study ML projects in open source, but usually (a) only focus on one or two specific examples or (b) use a dataset full of notebooks, research projects, homework solutions, and demos, which are not representative of real-world industrial ML products. While open source is useful for studying ML libraries and notebooks [26]–[28], researchers struggle to find good examples of open-source *ML products*: Several papers prominently highlight *FaceSwap* [29] as an active end-user open-source ML product and study it in depth, but it is usually the only clear example of an ML product ever identified or analyzed [30], [31]. Several papers [32]–[34] rely on a dataset of 4,500 projects labeled as "ML applied" [35], but closer inspection reveals that most of these projects are research notebooks, tutorial-style projects, and toy projects – not a promising dataset for those interested in studying ML products.

We have two goals for this paper, (a) to identify a corpus of ML products in open source, beyond just FaceSwap, and (b) to analyze the curated dataset to answer research questions of interest to the community. The first goal turned out to be surprisingly difficult due to the abundance of open-source ML projects that are not ML products, making the keyword search approach used in related studies ineffective. In response, we designed a tailored pipeline, strategized specifically for finding *ML products*. We identified and manually verified a total of 262 repositories [35]. While this is a smaller corpus than past datasets on ML projects, it provides a considerable number of open-source projects that build an end-user product around a model, have a development history, and are fully transparent, providing opportunities not achievable with interviews, surveys, and even industry collaborations.

Second, we analyze our dataset to answer existing research questions for which open-source software might provide useful insights. Instead of a shallow quantitative analysis of all the ML products, we conducted an in-depth analysis of a 30-product sample (Table I) and reported 21 findings around six research questions related to collaboration, architecture, process, testing, operations, and responsible AI. Among others, our findings reveal (a) often limited visible involvement of data scientists in developing open-source ML products, and a lack of clear boundaries between responsibilities for ML and non-ML code, (b) diverse architectural choices on incorporating often multiple models into products, and (c) rare use of industry best practices, such as pipeline automation, model evaluation, and monitoring. While our findings suggest that open-source ML products often mirror practices reported from startup-style ML products in industry, we also find a wide range of practices and products in this dataset. Our dataset offers ample research and educational opportunities, such as developing tools to assist teams that lack access to data scientists, tools and patterns to address the unexpectedly low modularity between ML and non-ML code, and open-source-specific innovations for privacy-preserving telemetry.

To summarize, the primary contributions of this paper are (a) an open-source dataset of 262 ML products, (b) a novel search strategy to identify the ML products from GitHub, and (c) 21 findings around six broad research questions, characterizing the nature of ML products in the dataset.

II. Defining ML Products

Throughout the paper, we use the term *ML product* to describe software products for end-users that contain ML components. We explicitly distinguish ML products from other ML-related *projects* and artifacts, such as notebooks and models. Note that terminology in this field is not standardized or consistent, as practitioners and researchers may refer to the libraries that train models (e.g., TensorFlow), the code to train models (e.g., in a notebook), the deployed models (e.g., GPT-3), or the products around those models (e.g., FaceSwap) by names such as ML systems, ML projects, or ML applications. Our notion of *ML product* considers the entire software system including non-ML components, in line with past research that used terms like ML-enabled systems [1], [36], or ML systems [5], [37].

During our research, we needed a clear definition for what we consider an *ML product* (especially because we had to classify thousands of repositories). In line with *interactive systems* in human-computer interaction [38], [39], and *products* in the context of product management [40], [41], and after iteratively reviewing hundreds of projects (cf. Sec. IV), we define *ML product* as follows:

> **ML Product:** A machine-learning product is a software project (a) **for end-users** that (b) contains one or more **machine-learning components**.
>
> To be considered *for end-users*, the project must have a *clear purpose* and a *clear target audience*. The purpose can be fun and the audience can be "everybody." The software must be *complete, usable*, *polished, and documented* (e.g., install and usage instructions) to the level typically expected by the target audience. The product needs to use at least one machine-learned model that provides major or minor functionality in the software. The model can be developed from scratch or called using an existing library or API.

For contrast, we define ML libraries and ML projects:

ML Library: Libraries, frameworks, or APIs that are used to perform ML tasks, such as TensorFlow and DVC[1].

[1] https://dvc.org/

ML Project: ML Project represents any software project that integrates some form of ML functionality or code. Examples include notebooks, research artifacts associated with a paper, and course homework. All ML products are ML projects, but most ML projects are not ML products.

Scoping. We exclude ML products targeting software developers and data scientists as end users from our corpus, such as code-completion tools. These users have more technical expertise and may interact with products through different interfaces, sometimes blurring the line between product and project. Researchers interested in ML products for developers could repeat our search process with a wider scope.

III. Existing Research and Limitations

Building ML products is challenging and requires engineering beyond the model and ML pipeline. Many researchers have studied the challenges that practitioners face when turning an ML model or prototype into a product. We recently collected challenges from 50 papers that surveyed and interviewed practitioners regarding the software engineering challenges faced when building ML products [42]. These papers illustrated numerous challenges, such as architectural issues due to lack of modularity in ML code increasing design complexity [5], [43] and team collaboration hindered by the absence of necessary skills [1], [44]. While these studies with practitioners provide a high-level understanding of the problems, they often do not offer sufficient details or access to design-specific interventions.

Researcher tradeoff between internal and external validity for studying ML products. For studying ML products (not just models and ML projects), researchers adopted different research designs. Some conducted interviews, e.g., [1], [5], [6], [43], while others focused on surveying practitioners at scale, e.g., [7], [9], [45]. While these studies provide a broad sense of the challenges (maximizing external validity), they rely on self-reported data without access to artifacts. There are also ethnographic studies [12], [13], industrial case studies [14], [46], and experience reports [10], [11] – these can yield a deeper understanding of specific cases (maximizing internal validity), but at the cost of generalizability as they are usually based on a single case (a common tradeoff [47]). Access to an open-source dataset of ML products can provide new opportunities, enabling researchers to validate reported challenges on tangible data, devise solutions, and evaluate interventions across a larger number of cases.

Academic interventions exist for models and pipelines where plenty of notebooks and libraries exist in open source to study, but not for *product-level* problems to which academics hardly have access. Academics can design and evaluate solutions when adequate data is available, such as the millions of notebooks on GitHub [27], [28], for example supporting studies and evaluations of interventions on dependency management, documentation generation, and collaboration practices in notebooks [27], [48], [49]. There are also many solutions for ML-related components such as data validation [50], [51], training data creation [52], [53], model building [54], [55], and fairness assessment [56], [57]. However, without access to study ML products, it is challenging to design and rigorously evaluate interventions at the product level (incl. architecture, collaboration, and documentation).

Open-source datasets exist for ML projects, but those are not representative of *ML products*, and not suitable for answering research questions related to ML product development. Several papers introduce datasets of ML *projects* on GitHub (not necessarily ML *products*). For instance, Gonzalez et al. [35] collected over 4500 "applied AI and ML" repositories. The quality of this dataset was criticized for inclusion of toy projects by Rzig et al. [30], who then curated a new dataset with 2915 ML projects for studying the adoption of continuous integration (CI) practices. Similar datasets of ML projects were curated by van Oort et al. [58] and Tang et al. [59] for purposes such as discovering the prevalence of code smells and refactoring practices in ML projects. Widyasari et al. [60] shared a manually-labeled dataset of 572 "engineered ML projects," referring to the need for a higher-quality dataset that should include engineering practices. While all these datasets can provide various insights about *ML projects*, they are not promising for studying concerns related to developing ML *products* (we analyzed all of them to find only four projects that we consider as products). We argue that insights about architecture [32], technical debt [34], and differences in addressing ML and non-ML issue reports [33] from studying ML projects might not well generalize to ML products.

Despite curating many datasets of ML-related projects presented above, researchers have not succeeded in finding ML products in open source, apart from one frequently highlighted example, FaceSwap [29]. Closest to our work, Wan et al. curate a dataset of ML products where a model is used in the context of some non-ML code (in their case to test the code *using* the model). Unfortunately, all their projects are toy projects, such as a 0-starred fire-alarm "application" with three commits that uses an object detection model to get object labels from an input photo and prints "alarm" if it detects the keyword "fire" in the label [61].

Existing datasets of ML projects and toy examples limit the generalizability of research to real-world industry-style *ML products*. Even worse, if a reader is not aware of the types of projects in a dataset (which is not obvious when the example provided is FaceSwap), it is easy to incorrectly generalize findings to *ML products*. Thus, there is value in a dataset specifically dedicated to ML products, which motivates us to curate and study such a dataset.

IV. Contribution A: Curating the Dataset

Identifying open-source ML products was surprisingly difficult and turned into a research project in itself. Searching with keywords like "machine learning" in READMEs, as in prior work collecting open-source ML projects [35], does not work because (a) the vast number of ML projects (libraries, notebooks, research experiments, demos) is much larger than the much smaller number of ML products and (b) ML products

do not always explicitly advertise their use of machine learning, especially when used for smaller optional features. For example, only one of the top 500 search results on GitHub for "machine learning" is an ML product and 13 of our 30 analyzed ML products do not mention machine learning in their README, such as the video-conferencing application (P29) which uses facial expression detection as an add-on.

Instead, we explored and iteratively refined a new search strategy combining domain knowledge, code search, and manual analysis in a process that is specifically designed to scale to search across all of GitHub. In a nutshell, our approach is based on the following insights:

- Targeting end users, ML products have a user interface (mobile, web, desktop, command line), whereas most other ML projects do not. We rely on code search to identify code relating to user interfaces.
- ML is used in products usually through a small number of libraries and APIs, whether to train a custom model, to load a serialized model, or to call a remote API service. We rely on code search to identify the use of ML in implementations.
- The final distinction between ML products and ML projects requires human judgment (all our attempts at automation yielded poor accuracy). We develop heuristics to prioritize which projects to analyze to manage scarce resources for manual analysis.
- Code search at the scale of GitHub is challenging. We carefully design a multi-step pipeline that incrementally reduces the search space, eliminating many projects that are not ML products with cheaper analyses before more expensive analysis steps are required.

Each insight makes assumptions that enable the search to scale and find relevant ML products, but each assumption may lose some ML products that do not meet them, such as user interface mechanisms not captured (e.g., game engines) and models not detected (e.g., custom k-nn implementations). Our approach cannot ensure finding an exhaustive list of all ML products – it is a best-effort attempt to collect as many ML products as possible with reasonable resources, in the face of a very difficult search challenge (see limitations below).

A. Search Space and Scope

We search for ML products on GitHub. GitHub is by far the most popular platform for open-source projects, whereas more specialized platforms such as Hugging Face only host ML models. We only include popular project repositories (over 50 stars) that have been maintained recently (updated after 2019-01-01), and that are documented in English – constraints that are common in open-source research. We restrict our analysis to desktop and web applications written in Javascript, Python, Java, and C# (most popular languages for such applications [62], [63]) in addition to mobile apps for Android and iOS.

B. Search Pipeline

To scale the search, we proceed in five steps, with increasing per-project analysis cost in each step.

1. API search. We start with a very scalable step to retrieve a vast overapproximation of candidate projects with the GitHub Search API. We retrieve all GitHub repositories using any of the four programming languages as the primary language and all repositories matching the keywords "android" or "ios." We additionally restrict the search to stars and commit date, as mentioned above. Where necessary, we partition the search space by date to overcome GitHub's maximum of 1000 search results. At this stage, we identified 430,902 candidate repositories (cf. Table II).

2. Metadata and README filter. We retrieve each candidate project's README and GitHub metadata (including "about" description and tagged topics) through the GitHub API. We exclude obvious non-product repositories by matching keywords such as "framework," "tutorial," and "demo" in the description or README. In line with similar efforts, we remove archived and deprecated repositories (e.g., keywords "deprecated" or "obsolete"), forks, and repositories with non-English descriptions (using an off-the-shelf model [64]). We report the exact filters in the appendix [65]. We manually validated a random set of 100 filtered projects finding no incorrectly filtered projects. A total of 300,508 repositories remained after applying this filter.

3. Product filter. To detect user interfaces, we rely on code search, performed locally after cloning each candidate repository. We curated a list of code fragments indicative of 130 common frameworks for user interfaces, such as "com.android.application" in a gradle.build file for Android mobile applications, "import javax.swing" for Java desktop applications, and "from flask import Flask" for web applications in Python (see appendix for details [65]). We remove repositories that do not contain any of these code fragments, leaving us with 26,386 potential products for further analysis.

4. ML filter. To identify the use of machine learning, we again rely on code search, based on curated lists of code fragments indicative of ML libraries and APIs. We count occurrences of calls to any of 99 ML libraries or APIs (e.g., "import caffe") and of serialized models (e.g., files with .tflite, and .mlmodel extensions). In addition, as a noisy last resort, we count occurrences of 20 ML keywords, such as "machine learning" and "NLP" in any source or text files (including comments and documentation) to catch less common libraries and custom implementations. At this point, 11,257 projects pass at least one ML-related filter.

5. Manual inspection. The final step with by far the highest per-repository cost is to manually validate whether a repository is an ML product. One or more authors with extensive expertise in ML products inspected the repository, its description, and (when needed) its code to judge whether the repository is indeed an ML product – this typically took 30 seconds to 20 minutes per repository. Our definition of ML product in Sec. 2 is the result of multiple iterations and refinement, for example, establishing requirements for purpose and documentation, for which we discussed 272 early-inspected projects as a group

TABLE II: Number of Retrieved Projects after Each Step

	Mobile App		Desktop App				Web App			Total
	android	**iOS**	**js**	**py**	**java**	**C#**	**js**	**py**	**java**	
GitHub Search	12,044	10,969	72,793	67,626	55,892	15,267	72,793	67,626	55,892	430,902
Initial Filter	3,358	4,055	83,777	36,396	22,802	7,145	83,777	36,396	22,802	300,508
Product Filter	2,296	2,801	1,100	1,663	1,909	2,590	3,025	8,747	2,255	26,386
Manual Check*	33	14	19	43	17	12	42	104	5	****262**

*Based on the ML clue counts, we inspected around 4k projects manually. **Removing duplicates.

(of which we considered 94 to be ML products) to arrive at a stable definition which provided us with a high inter-rater reliability (n=40, kappa=0.77). A few repositories near the decision boundary were discussed by all authors until a unanimous consensus was reached. We inspected about 4,000 of the 11,257 remaining repositories, prioritizing our resources based on match counts for our ML filters, stratified product category, language, and ML filter. In each strata, we stopped when we reached 30 consecutive false positive repositories, for example after inspecting 216 Android mobile apps.

C. Limitations and Threats to Validity

To make the search feasible we had to make various compromises, arriving at the described design. Given the various heuristics, our approach represents a best-effort attempt and cannot claim producing an exhaustive or complete list of ML products. As discussed, we may have missed ML products in other languages, using other GUI frameworks, or less common ML libraries. Additionally, our approach involved manual inspection, which, despite best efforts, opens the possibility of human error and subjectivity.

Our search heuristics prioritize false positives over false negatives, and we designed our approach accepting low precision (discarding many repositories in the last manual validation step) to ensure high recall. While we would have preferred to formally evaluate recall (i.e., whether we missed any ML products) by comparing our dataset against any existing ground-truth dataset of ML products, such a dataset does not exist. As a substitute, we attempted to collect ML products independently by seeking input from industry practitioners through platforms like Quora, Reddit, LinkedIn, Twitter, and a 32k-member Slack channel in the field of data science; but aside from numerous replies expressing interest in our dataset, we only received suggestions for two repositories, both of which we determined not to be ML products according to our definition. Additionally, we compared our dataset to other existing datasets of ML projects [26], [35], [59], [60] but did not find any additional repositories that satisfy our definition of ML products in those datasets. In fact, those datasets only contained a total of four ML products, all of which we detected in our dataset. While all this raises our confidence, we cannot formally assess recall.

D. The Open-Source ML Product Dataset

In total, we found 262[2] ML products (cf. Table II, full dataset in appendix [65]). The average ML product in our corpus has 1495 stars, 28 contributors, and is 325MB in size. Over half of the ML products are written in Python; most are web applications.

The dataset comprises a diverse range of products, some of which have a significantly larger number of users and a more professional look than others. For instance, Seek (P2 in Table I) is a mobile app for identifying plants and animals using image recognition, downloaded over 1 million times and reviewed by over 38k users with robust support from the established iNaturalist community, who maintains a dedicated website and continuously improves and maintains the app. In contrast, NotionAI MyMind (P5), an Android app developed by a single contributor, uses an ML classifier to automatically tag images and articles, with a simple user interface, rare updates, and under 5,000 downloads from the Play Store. Approximately half of the products in our dataset have a professional presentation like Seek; those generally have more stars and a larger codebase. The others seem to be personal-interest projects released as a product.

V. Contribution B: Learning from the Dataset

We created this dataset due to limited access to industry products, which hinders research advancements and education in the field. Now that we have this dataset, we can finally attempt to answer numerous research questions about ML products that have accumulated in many past studies with open-source products, where previously we had to rely on interviews or experience reports from industry practitioners. Given the breadth of topics, we cannot cover everything in a single study. In this paper, we explore a wide range of topics rather than going into depth on a single one, to *contribute new knowledge* and achieve two secondary objectives: **(1)** *Characterize the dataset:* Our analysis with broad research questions will implicitly characterize the products in our dataset and enable other researchers to effectively use it and interpret derived findings (cf. limitations of existing work in Section III) – this also helps to explore how similar the ML products in our dataset are to ML products described in interviews and experience reports. **(2)** *Identify when deeper analysis of the dataset is feasible and promising:* Our research questions from different topics, such as collaboration, architecture, and

[2]Removing duplicates, such as same repositories for Android and iOS apps.

development process will identify what kind of questions are worth going into deeper before committing resources to in-depth analyses for individual research questions.

A. Research Method

First, we curated a list of research questions relevant to the study and designed qualitative and quantitative strategies to answer them. Given the novelty of the dataset and questions, we heavily rely on qualitative analysis involving substantial manual effort. Therefore, we decided it would be more manageable to analyze a sample of 30 products from the dataset.

Deriving Research Questions. We selected research questions that are not only of interest to the research community but can also be feasibly answered by analyzing open-source ML products (e.g., we did not find artifacts that would allow us to answer *"How do data scientists elicit, document, and analyze requirements for ML systems?"*). For this selection, we employed a two-step process.

First, we explored the existing literature to identify topics that are of interest to researchers in the field, such as the challenges faced by practitioners building ML products, collected in our recent meta-survey of interviews and surveys [42]. We identified numerous topics of interest, such as collaboration, architecture, process, quality assurance, MLOps, and responsible AI.

Second, we examined our dataset of ML products to identify potential research questions (RQs) that could plausibly be answered with open-source data. We explored 15 randomly selected ML products qualitatively, taking multiple pages of notes for each product. We immersed ourselves in the source code, documentation, contributor profiles, issues, and any other available information provided on associated websites; we identified the ML and non-ML components to familiarize us with common structures. This gave us a sense of what kind of questions can be reasonably answered.

In the end, we selected six questions spanning the entire life cycle: • **RQ #1 (Collaboration):** How interdisciplinary are open-source ML product teams and how do they divide their work? [1], [66] • **RQ #2 (Architecture):** How are open-source ML products architected to incorporate models? [5], [32] • **RQ #3 (Process):** What model-product development trajectory do open-source ML products follow? [4], [67] • **RQ #4 (Testing):** What and how are the open-source ML products and their parts tested? [37], [68] • **RQ #5 (Operations):** How are open-source ML products designed for operation? [44], [69] • **RQ #6 (Responsible AI):** What responsible AI practices are used in open-source ML products? [70], [71]

While each identified topic could warrant a dedicated study and deeper analysis, here we provide initial answers for each and explore opportunities for future research.

Analysis and Synthesis. Without existing established measures, we found a manual, mostly-qualitative analysis to be more responsive and effective than a narrow quantitative analysis at scale [72], [73]. To find answers to the RQs, the ML products required an in-depth examination of the code, analysis of contributor activities, and thorough inspection of related documents. While we automated some measures, such as contributor percentages and modularity scores, designing them also required initial manual investigation and manual classification of ML components. This made the process quite labor-intensive, requiring 10-15 hours per product. Consequently, we analyzed a sample rather than every product in the dataset.

Sampling: We analyzed the 30 products shown in Table I (11.5% of the dataset), which was manageable for our manual analysis. Rather than attempting statistical generalizations, our goal was to gain rich insights from the dataset. Thus, we aimed to sample a diverse range of products using case-study research logic [74], [75]. We used *information-oriented selection* to select popular and large products (*extreme/deviant cases*), because we expect them have a richer history to analyze, and we included a random selection of other products for the *average cases*. To represent different kinds of products, we stratified selection across the three genres: mobile, desktop, and web. Specifically, we select two products with the most stars per genre (P8, P9, P16, P17, P21, P28), two products with the most contributors per genre (P3, P6, P14, P19, P24, P25), the product with the largest code size per genre (P1, P11, P29), and five randomly selected products per genre. The sample set of ML Products vary in terms of stars (avg. 5k, min. 63, max. 42k), no of contributors (avg. 82, min. 1, max. 374), and LOC (avg. 564k, min. 1106, max. 2745k), as shown in Table I and appendix [65].

Analyzing Products and Card Sorting: We conducted a comprehensive qualitative analysis of the sampled products. We describe the specific analysis steps separately for each research question below, but generally, we follow the strategy: two researchers carefully examined the GitHub repositories, addressing each research question individually for each product, involving tasks, such as reading documentation, identifying ML and non-ML components in the source code, measuring modularity, examining contributor profiles, analyzing commit history, and reviewing issues. The entire research team regularly met to discuss, clarify understanding and resolve disagreements, and organize findings. To organize and find patterns among the products, we performed card sorting [76]. Each product was represented by a card for each RQ, describing our findings for that particular RQ, and we iteratively grouped these cards to identify patterns within RQs. Additionally, we searched for associations across patterns from different RQs. We share analysis artifacts, including a Miro board and spreadsheets, in the appendix [65].

Threats to Credibility and Validity. Despite following best practices for qualitative research as we discussed, this part of the research shares common threats encountered in qualitative research [72], [73]. Given the small sample size and sampling strategy, statistical generalization is not suitable and not advised. Readers should compare this to a study with 30 interviews. While we followed standard practices for coding and memoing during the analysis of the products, we cannot

completely eliminate biases introduced by the researchers. In addition, we only access public information and do not have access to offline activities – hence, our findings should be interpreted accordingly.

RQ #1: How interdisciplinary are open-source ML product teams and how do they divide work?

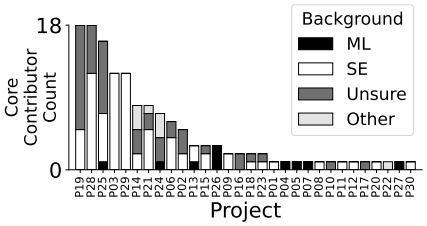
Interdisciplinary collaboration is difficult, as has also been found in building ML products [1], [66]. The transparency of open-source development allows researchers to study many aspects of collaboration, as demonstrated by numerous past studies on team collaboration, pull requests, and diversity, e.g., [77]–[79]. To understand *interdisciplinary* collaboration in open-source ML products, we explore team composition in terms of the number of contributors and their backgrounds, who works on ML and non-ML code of the product, and how tangled ML and non-ML code are in terms of co-changes. The findings can help study existing challenges such as siloed development and guide further studies in collaboration.

Method. To analyze contributor backgrounds and numbers (*Findings 1-2*), we collected contribution data from GitHub and identified the *core* contributors as those collectively responsible for 80% of all commits (in line with past work [80], [81]). We manually classified each contributor's background as *SE-focused, ML-focused*, or *other* (e.g., physics, finance), based on public self-description, professional title, and education history as found on their GitHub profile, LinkedIn profile, and personal, project, or company websites, if available. If the classification was not obvious (e.g., because of limited public information) we classify their background as *"unsure."*

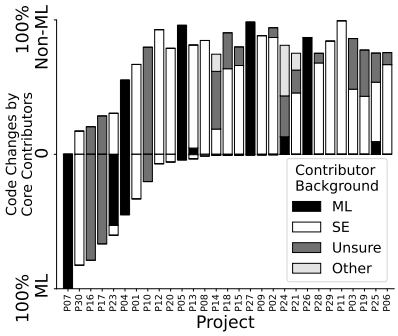
To study how developers from SE and ML backgrounds contribute to ML and non-ML code (*Findings 3-4*), we separated the ML and non-ML code after excluding documentation and binaries: We manually categorized code associated with model training, prediction, and pipeline as ML-related, while all other software infrastructure and graphical user interface (GUI) code fell under non-ML, typically at the granularity of files. Finally, we automatically analyze the commit history to attribute code changes to contributors.

To analyze the coupling of ML and non-ML code (*Finding 5*), we analyze co-edits in the product history, known as *logical coupling* [82], [83]. Specifically, we compute a *relative coupling index* that indicates whether the ML and non-ML parts are more or less coupled than would be expected if all changes were randomly distributed across files. A low *relative coupling index* indicates that changes are typically isolated to only ML code or only non-ML code, whereas a high *relative coupling index* indicates that ML and non-ML code are often changed together, signaling low modularity. To compute *relative coupling index*, we compute coupling using *evolutionary coupling index (ECI) [82]* and then divide *ECI* by the probability of coupling of random edits to normalize the effect of size, as ML and non-ML code size differs significantly (the average product has 971k LOC of non-ML code and 23k LOC of ML code).

Finding 1: Most of the core contributors are software engineers (74/92). Finding 2: Many products have a single core contributor (13/30). In our sample of 30 products, we identified 140 core contributors, among whom we could classify 92. Among the 92, we found 74 contributors as *SE-focused and* 10 contributors (spread across eight products) as *ML-focused*. For 16 products where we could classify all core contributors, many (9/16) had exclusively *SE-focused* contributors. We found a single contributor (in P24) who self-identified to be an expert in both SE and ML.

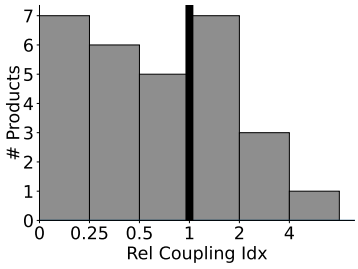


Finding 3: There is little evidence of clear silos, with core contributors commonly committing to both ML and non-ML code, regardless of background. Finding 4: Team responsibilities are rarely assigned and recognizable in the commit history. In contrast to the widely reported problem of *siloing* in industry teams [1], we do not find a clear delineation by background of who contributes to the ML and non-ML code. We often find contributors of either background working on both parts.



Only five products (P13, P21, P24, P25, P29) publicly documented team structures with assigned roles and responsibilities for team members, but even then the assigned responsibilities do not always reflect the commit activities. For instance, even though P29 has an explicit data team, we do not find commits from the data team to the ML repository, but *SE-focused* contributors change both the ML and non-ML code. We conjecture that some offline collaboration is not visible in the open-source repository.

Finding 5: ML and non-ML code are often changed together, indicating low levels of modularity. While models are usually assumed to be modular components in a system, our analysis reveals that many products (12/30) in our sample exhibit frequent co-changes of ML and non-ML code. For example, P27 has a very high *relative coupling index* between ML and non-ML code – we found that this product has a custom script for training a speech recognition model,



where the ML code directly updates the user interface (UI) button based on the prediction result; any modification to the UI properties requires an update to the model script to accommodate the change, causing frequent co-changes of the UI files and the model script. Conversely, although we indeed found many products (18/30) that have coupling lower than random, it is not as low as could be expected from fully modular components of a product.

Discussion (open source versus industry): Open source ML products mirror the startup style of development more than big tech projects. In line with general trends in open source [84], [85], we find relatively small teams developing ML products in our sample, in contrast to often large teams reported to build ML products in big tech companies [43], [86]. Our open-source ML products are closer to the average team size of 2-5 members in startups [87]. While prior studies report misaligned responsibilities that do not reflect developers' abilities or preferences across all kinds of organizations building ML products [1], [66], [88], the fluent and broad responsibilities and collective code-ownership resemble characteristics commonly seen in startups. Overall, the open-source ML products seem more reflective of activities in the vast majority of ML projects outside of big tech organizations, that have their own distinct and often understudied challenges [1], [2], [89].

Implication 1: Researchers should study the challenges of teams that do not have access to data scientists and explore providing assistance. Most studies on ML development focus on the challenges of data scientists, perceived as the dominant or novel role. Open-source ML products seem to be dominated by software engineers, who adopt ML tools, often with limited apparent participation from data scientists. Past interview studies have already established pitfalls of software engineers adopting ML without explicit training [1], such as inadequate feature engineering and insufficient evaluation. Open-source ML products provide an opportunity to study such problems in public artifacts. In addition, ML products developed without dedicated data scientists are likely common also in industry, outside of big tech, and likely to become more common as data science becomes more accessible (e.g., with AutoML and prompt engineering) – researchers should explore how to support software engineers with limited data-science expertise in building ML products responsibly, for example, through analysis, automation, and smart assistants. Recent tools for detecting data smells [90] and data leakage [91] and for anticipating fairness issues [92] provide encouraging starting points. Conversely, a few ML products in our sample are developed by data scientists without software engineers – such cases are better researched [1], [93], but can equally benefit from further studies and support.

Implication 2: Researchers should investigate sources of non-modularity and develop tools and guidance. While interactions among multiple models are a well-known problem (*"changing anything changes everything")* [3], [94], [95], models are usually considered natural modules in a software design with clear and simple interfaces [96], [97]. Yet, we found surprisingly frequent co-changes of ML and non-ML code. Research should explore the sources of this non-modularity and have a unique opportunity to do so with our dataset. Research should identify or create design strategies to isolate change, possibly coded as design patterns (current ML-related design patterns rarely consider the interaction of ML and non-ML code [98], [99]), to guide practitioners toward more modular designs. Positive examples in the dataset could serve as illustrations in educational materials.

RQ #2: How are open-source ML products architected to incorporate models?

Researchers have highlighted how ML can influence the architecture of software products [5], [43]. To comprehensively understand the product structures and the incorporated models, we explore architecturally relevant aspects such as model type, usage, importance, integration of multiple models, pipeline automation, documentation, and big data infrastructure.

Method. To understand the overall structure of the model and product, we conducted a comprehensive manual analysis of the ML and non-ML code in the repositories. We analyzed the code with a focus on the following artifacts: code structure and data flow as it pertains to models – identifying how the models are created, where and how they are called, and how the model predictions are processed and used. We also reviewed their documentation, relevant blogs and forums, associated web pages, and related repositories under the same personal or organizational account. We then sorted our findings and grouped those into categories, using card sorting techniques [76], guidance from previous research, and domain knowledge from our research team.

Finding 6: About half of the products rely exclusively on third-party ML models (13/30). We identified 15 products that use *third-party models* via libraries (e.g., Tesseract OCR), external APIs (e.g., ClarifyAI), or load pre-trained model files from a remote repository. In contrast, 17 products *self-train models*. Two products use both *third-party* or *self-trained models* (P12, P30). For instance, the optical music recognition application, P12, uses a self-trained model to classify music symbols and an existing OCR library for classifying text.

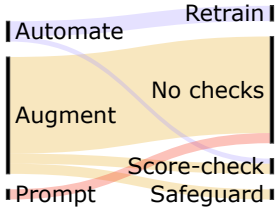
Finding 7: The importance of the ML models to the product varies, with about half using them as optional functionality only (13/30). We found the importance of ML models to vary considerably across different products. The model is the *core* functionality in 11 products, as there would be no product without the model (e.g., the OCR model in the OCR scanner app P1). There are 6 products that may still provide value without the model, but the model is a *significant* functionality (e.g., the OCR model in video subtitle editor P15 that could potentially operate on manual inputs). In 13 products, the model provides *optional* functionality, serving as a nice-to-have add-on (e.g., facial expression recognition in video conferencing app P28). Whether a *third-party* model is

used (*Finding 6*) is not necessarily associated with the model's importance: We found products investing substantial effort in self-trained models for *optional* functionality (e.g., P6, P26, P29) and products relying on a third-party model for *core* functionality (e.g., P1, P12, P13). Products with models as *core* tend to be smaller and have fewer contributors (avg. 112k LOC, 1.4 contributors), while *optional* models are often added to larger products with more contributors (avg. 754k LOC, 8.1 contributors).

Finding 8: Automation using model predictions is uncommon (5/30), with most products keeping humans in the loop. A central question in human-AI design is how to use or present model predictions and whether and how to keep humans in the loop [97], [100]. We find only five products that use model predictions to fully *automate* actions (e.g., keyword spotting app P27 executes gameplay instructions based on recognized voice commands). Two products *prompt* users to confirm an action (e.g., deepfake software P17 asks for confirmations on image previews between each processing stage). Most products (23) in our sample merely *display* predictions, leaving decisions about actions entirely to users (e.g., trading app P21 graphically presents investment predictions).

Finding 9: Most products use raw model predictions without any post-processing (21/30). Finding 10: Products that automate actions are more likely to further process model predictions. Products may check, process, and integrate model predictions in many ways, some now encoded as patterns, such as *two-phased predictions* for resilient serving of models [8], [98]. However, most analyzed products (21/30) trust model predictions and display them without any further processing. Only two products incorporate additional architectural tactics around model predictions: Plant identifier app P2 uses a two-phase prediction system, combining a local model with an online model for low-confidence cases (known as *two-phase prediction pattern* [98]) and subtitle editor P15 performs extensive checks on texts predicted via optical character recognition and language translation before presenting them. In addition, three products incorporate a confidence score threshold to filter low confidence predictions (P9, P27, P29) and another three offer a retraining option for the model if performance proves unsatisfactory (P5, P12, P28). Interestingly, P11 uses machine learning to check the results from a non-ML API. The few products that automated decisions based on model predictions (Finding 8) process predictions further, by offering retraining mechanisms (3/5) and confidence checks (1/5).

Finding 11: Many products use multiple models (18/30), though those models are mostly independent (11/18). Interactions among multiple models is a frequently raised challenge in industry, where a minor change in one model can trigger cascading changes across the product [3], [94], [95], [97]. While 18 products use multiple models, those perform independent tasks in 11 products[3]: 7 products use models for *separate functions* unrelated to each other (e.g., P15 uses one model for OCR and another for speech-to-text) and 7 products provide *alternative* models for the same function (e.g., P26 provides a choice between two clustering models). Five products *sequentially compose* models [97] (e.g., P12 passes text recognized by an OCR model to an entity recognition model). Two products use models for *collective decision-making* (e.g., P9 combines multiple classifiers to generate personalized news feeds).



Finding 12: Pipeline automation is not common in open-source ML products. Switching from a static mindset and notebooks to pipeline automation is a commonly reported challenge [1], [101]. Among the 17 products that use self-trained models (*Finding 7*), training is often not automated. We did not find any model training pipeline for four products (P2, P18, P22, P30; we cannot tell if training happens offline or in a private repository). Four products (P6, P10, P12, P16) require manual execution of sequential training steps; two products automate only data retrieval (P11, P24)[3]. Four products (P9, P15, P27, P29) have GUI-integrated training pipelines that can be separately activated via GUI actions. Only four products (P14, P21, P23, P26) feature fully automated training pipelines to consistently fetch the latest data and deploy updated models.

Finding 13: We do not find much effort on data or model documentation. Both industry and academia view model and data documentation as important for, among others, collaboration, accountability, and reuse [102]–[105], but adoption in industry is rare and perceived as challenging [48], [102]. In our sample, the 17 products using self-trained models (*Finding 7*) provided minimal and mostly scattered documentation for models and data, if any. Regarding *model documentation*, only one product, P29, provides high-quality model documentation (in the form of a model card [106]). Other products have at most brief instructions for using the model API or descriptions of the model architecture for the data scientists. *Data documentation* was mostly limited to presenting a data schema, occasionally mentioning the volume of the training data, or simply providing a link to their data sources. We found no use of datasheets [107] or similar templates.

Finding 14: Most products do not use big data infrastructure (23/30). Scalability is reported as a common, important architectural challenge for ML products, for handling large datasets and distributing expensive training and inference jobs, resulting in frequent reliance on big data infrastructure. However, we did not find the use of local or self-hosted big data infrastructure (such as Hadoop and Spark) in any of our sample ML products. Seven products contain code related to cloud services for storage, computing, monitoring, and search (e.g., Amazon S3, EC2, CloudWatch, and Elastic Cloud).

Discussion (open source versus industry): Open-source ML products share many of the development decisions and

[3]Numbers do not add up, as some products fall in multiple categories.

challenges discussed in industry studies. In line with the industry trends, many open-source projects leverage third-party models rather than building models from scratch, a pragmatic choice given the cost, time, and skill requirements involved in developing models with limited resources [89], [108], [109]. We find models used for various tasks with varying levels of importance in a product, reflecting the diversity of the ML products. Open-source ML products with models as *core* more resemble startup-style projects, whereas attempts to integrate models as enhancements to existing products are found throughout the industry, including many established corporations. While there are large variations within our dataset, open-source ML products tend to lean toward the less complex end of reported ML products in the industry, with simpler architectures with few models, limited automation, and less need for massive scale. The lack of pipeline thinking, overly trusting model predictions, and poor documentation mirror practices repeatedly criticized in industry projects [42], [69], [102], and while the more experienced and well-resourced companies work toward better practices [68], [86], [98], [106] those challenges are still common in many newer and smaller organizations.

Implication 3: Researchers should study patterns and test interventions for different architectural choices. While researchers and practitioners argue for the need to implement safeguards [110], [111], prepare for the evolution of third-party models [108], [112], design effective and safe human-AI interaction models [113], and integrate multiple models [3], [95], researchers rarely have access to enough products to detect patterns and validate solutions on a range of systems. Even if some of the practices are rare in open source, our dataset has many and diverse projects to study existing patterns and to provide a testbed to evaluate the consequences of different design interventions, such as design patterns to isolate models or data-flow analyses to track whether and how multiple models in a product interact. It also provides opportunities for deeper investigation in certain aspects, such as employing firehouse studies [114] to interview developers when certain events occur. Additionally, it allows researchers to conduct longitudinal studies to understand the evolution of the team and the architecture over time. We have not seen any prior longitudinal studies of projects in this field (common in MSR-style research), likely due to a lack of access.

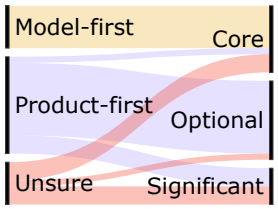
Implication 4: Educators should use open source to develop teaching materials. In a field where access to concrete implementations is scarce and educators often rely on demo projects or second-hand reports, open-source ML products can be a valuable educational resource to showcase system design strategies and challenges, whether as illustrations in lectures and blog posts, as foundations for homework assignments, or as in-depth case studies as in the *Architecture of Open Source Application* books [24], [115]. The dataset has sufficient variety to cover simpler projects suitable for beginners, as well as sophisticated products built by large teams to study architectural design decisions.

Implication 5: Companies, foundations, and governments should explore strategies to sustain model and big data infrastructure. Unlike revenue-generating commercial products, most open-source ML products in our sample (27/30) did not seek to monetize their products. The potential high recurring cost for model APIs and cloud computing may prevent open-source developers from scaling their products or from building certain products in the first place. Some open-source products may find a path to secure funding; for example, Seek by iNaturalist P2 was supported by various nonprofit foundations before establishing its own nonprofit with a seed grant. While companies, foundations, and governments often support open source (e.g., free hosting and CI on GitHub; Sovereign Tech Fund, NSF POSE), sustained support for model APIs and cloud computing is less common. Such support seems essential to encourage open-source innovations as alternatives to commercially dominated ML products.

RQ #3-6: Process, Testing, Operations, and Responsible AI

Due to the page limit, we only report brief findings from the remaining four research questions, but refer the interested reader to our appendix [65] for details on the methods, findings, and discussions. The following findings 15-16 relate to RQ #3, finding 17 to RQ #4, findings 18-20 to RQ #5, and finding 21 to RQ #6.

Finding 15: *Product-first development* (16/30) is more common than *model-first* (7/30). Finding 16: When the model is the *core* functionality, it is always developed first. In prior work, we found that some projects start with models and later build products around them (model first) whereas others adopt a product-first approach – each creating distinct challenges [1]. In our open-source sample, we observe a greater prevalence of the product-first trajectory, which may be attributed to most contributors being software engineers (*Finding 2*) and many products adding machine learning for optional functionality to existing products (*Finding 7*). Noticeably, products with models as *core* are always developed *model-first*. For example, deepfakes software P17, created the model first and added a GUI a year later to make the model accessible to end users.



Finding 17: Testing regular software functionality is common (23/30), model testing is notably scarce (8/30), and data validation is rare (2/30). Standard software testing practices are widespread, but model evaluation is less prevalent in our sample. Even among the eight products that included model evaluation scripts, three (P3, P9, P29) approached model testing like unit testing, asserting that predictions match expected values. The rare cases (P6, P21) in which data validation is conducted involve only minimal checks for schema and value ranges.

Finding 18: Only a few products (8/17) have mechanisms for evolving models. Of the 17 products with self-trained

models, five products (P9, P16, P23, P27, P30) offer users the option to retrain products at run time and three products (P14, P21, P26) continually retrain their models by fetching up-to-date data from their data sources. This aligns with our recent finding that many product teams have a static view of models [1].

Finding 19: Model monitoring is almost non-existent (1/30). Despite the heavy emphasis on observability in industry and academic literature for detecting failures and degradation [44], [116], 29 out of the 30 products do not collect telemetry and have no monitoring infrastructure. Only P21 incorporated telemetry for financial forecasting. In addition, P9 uses Amazon CloudWatch, but not for model monitoring.

Finding 20: MLOps tools are not used. We did not find use of any popular MLOps tools for tasks such as automating deployment, testing, monitoring, and data cataloging, in any of the products. Given that many of these products do not incorporate retraining mechanisms (*Finding 18*), they may have less need for MLOps automation.

Finding 21: Responsible AI practices (e.g., fairness, safety, security) are not apparent in open-source ML products. Despite significant attention in academia, we do not find adoption of any responsible AI practices in our sample. Only one product, P17, discusses ethical usage in their *README* [29], largely limited to disclaimers. Several products include privacy policies and disclaimers unrelated to ML. One educational product (P29) covers responsible AI practices as a subject.

Discussion (open source versus industry): In comparison to industry, open source showcases similar and more bad practices associated with both model and product evaluation and maintenance. Similar to our observations on architecture, open-source ML products exhibit many of the characteristics criticized in past research, with low adoption of tooling and interventions discussed by more experienced and well-resourced organizations – we find the same low adoption of model evaluation [117], data validation [118], monitoring [42], [89], and responsible AI practices [1], [48], [70]. The open-source ML products seem to reflect the practices of new organizations and smaller teams more than those of big tech organizations; they are likely reflective of the challenges that new teams will experience, especially teams dominated by software engineers.

Implication 6: Tool vendors have an opportunity to showcase the benefits of automation tooling. Rather than relying on testimonials and narrowly scoped tutorials, tool vendors such as those of MLOps tools can demonstrate their tools in forks of open-source ML products or can even work with open-source developers to integrate them into their products. For example, while researchers have found some public (mostly very small) projects using the versioning tool DVC [119], none of them were full ML products showcasing the integration of model and product versioning. Similar to Implication 5, tool vendors can provide resources to support open-source communities.

Implication 7: Research should explore open-source friendly monitoring approaches. Observability is a key focus of the MLOps community; many researchers and practitioners argue that the unreliable nature of ML and presence of data drift makes monitoring and testing in production crucial to responsible engineering [120], [121]. We conjecture that the privacy-conscious open-source culture and a lack of centralized infrastructure contribute to the minimal adoption of monitoring among open-source ML products. Researchers should explore privacy-preserving and community-operated monitoring solutions compatible with open-source values, ideally through co-design processes with open-source practitioners.

VI. Conclusion

We offer a dataset of 262 open-source ML products to facilitate research experiments that can benefit from access to the development history and artifacts of ML products, and report 21 findings and seven implications from six research questions. The dataset is a valuable educational resource for both academics and practitioners in ML product development, offering diverse study materials with both large and small ML products, and it also provides ample research opportunities as described.

Acknowledgment

The authors would like to thank Bogdan Vasilescu, Rohan Padhye, and Eunsuk Kang for helping with the framing, and their continuous suggestions and feedback. The author would also like to thank the industry practitioners who responded to the queries about ML products on Quora, Reddit, LinkedIn, Twitter, and Slack.

Kästner's, Nahar's, and Zhang's work was supported in part by the National Science Foundation (#2131477), Zhou's work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC, RGPIN2021-03538), and Lewis' work was funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center (DM24-1039).

References

[1] N. Nahar, S. Zhou, G. Lewis and C. Kästner, "Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process," in *Proc. 44th Int'l Conf. on Software Engineering*, 2022, pp. 413–425.

[2] A. Arpteg, B. Brinne, L. Crnkovic-Friis and J. Bosch, "Software Engineering Challenges of Deep Learning," in *Proc. 44th Euromicro Conf. on SEAA*, 2018, pp. 50–59.

[3] D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," *Adv. in Neu. Info. Proc. Sys.*, vol. 28, pp. 2503–2511, 2015.

[4] E. d. S. Nascimento et al., "Understanding Development Process of Machine Learning Systems: Challenges and Solutions," in *Proc. Int'l Symposium on ESEM*, 2019, pp. 1–6.

[5] G. A. Lewis, I. Ozkaya, and X. Xu, "Software Architecture Challenges for ML Systems," in *Proc. ICSME*, 2021, pp. 634–638.

[6] A. Vogelsang and M. Borg, "Requirements Engineering for Machine Learning: Perspectives from Data Scientists," in *Proc. 27th Int'l Requirements Engineering Conference Workshops*, 2019, pp. 245–251.

[7] F. Ishikawa and N. Yoshioka, "How do engineers perceive difficulties in engineering of machine-learning systems? - questionnaire survey," in *Proc. Joint 7th CESI and 6th SER&IP*, 2019, pp. 2–9.

[8] L. E. Lwakatare et al., "A taxonomy of software engineering challenges for machine learning systems: An empirical investigation," in *Proc. Int'l Conf. on Agile Software Development*, 2019, pp. 227–243.

[9] Z. Wan, X. Xia, D. Lo and G.C. Murphy, "How does Machine Learning Change Software Development Practices?," *IEEE Trans. Software Eng.*, pp. 1–1, 2019.

[10] L. Bernardi, T. Mavridis, and P. Estevez, "150 Successful Machine Learning Models: 6 Lessons Learned at Booking.com," in *Proc. 25th Int'l Conf. on ACM SIGKDD*, 2019, pp. 1743–1751.

[11] J. Lin and A. Kolcz, "Large-scale machine learning at twitter," in *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, 2012, pp. 793–804.

[12] S. Passi and P. Sengers, "Making data science systems work," *Big Data and Society*, vol. 7, no. 2, p. 205395172093960, 2020.

[13] S. Passi and S. J. Jackson, "Trust in Data Science: Collaboration, Translation, and Accountability in Corporate Data Science Projects," *Proc. of the ACM on HCI*, vol. 2, no. CSCW, pp. 1–28, 2018.

[14] M. S. Rahman et al., "Machine Learning Software Engineering in Practice: An Industrial Case Study," *arXiv [cs.SE]*, 2019.

[15] A. E. Hassan and T. Xie, "Software intelligence: the future of mining software engineering data," in *Proc. FSE/SDP workshop on Future of software engineering research*, 2010, pp. 161–166.

[16] A. E. Hassan, "The road ahead for Mining Software Repositories," in *Frontiers of Software Maintenance*, 2008, pp. 48–57.

[17] B. Vasilescu et al., "Quality and productivity outcomes relating to continuous integration in GitHub," in *Proc. 10th ACM Joint Meeting on ESEC/FSE*, 2015, pp. 805–816.

[18] G. Gousios, M. Pinzger, and A. van Deursen, "An exploratory study of the pull-based software development model," in *Proc. 36th Int'l Conf. on Software Engineering*, 2014, pp. 345–355.

[19] C. L. Goues, M. Pradel, and A. Roychoudhury, "Automated program repair," *Commun. ACM*, vol. 62, no. 12, pp. 56–65, 2019.

[20] A. B. Sánchez et al., "Mutation testing in the wild: findings from GitHub," *Empirical Software Engineering*, vol. 27, no. 6, p. 132, 2022.

[21] A. Marginean et al., "SapFix: Automated End-to-End Repair at Scale," in *Proc. 41st ICSE-SEIP*, 2019, pp. 269–278.

[22] G. Petrović and M. Ivanković, "State of mutation testing at google," in *Proc. 40th ICSE-SEIP*, 2018, pp. 163–171.

[23] D. M. C. Nascimento, C. F. Chavez and R. A. Bittencourt, "The Adoption of Open Source Projects in Engineering Education: A Real Software Development Experience," in *Proc. Conf. Frontiers in Education*, 2018, pp. 1–9.

[24] A. Brown and G. Wilson, *The Architecture of Open Source Applications*, vol. 1. Lulu.com, 2011.

[25] E. Gamma and K. Beck, *Contributing to Eclipse: Principles, Patterns, and Plug-ins*. Addison-Wesley Professional, 2004.

[26] M. Dilhara, A. Ketkar, and D. Dig, "Understanding Software-2.0: A Study of Machine Learning Library Usage and Evolution," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 4, pp. 1–42, 2021.

[27] J. F. Pimentel, L. Murta, V. Braganholo and J. Freire, "A large-scale study about quality and reproducibility of jupyter notebooks," in *Proc. 16th Int'l Conf. on MSR*, 2019.

[28] F. Psallidas et al., "Data Science Through the Looking Glass: Analysis of Millions of GitHub Notebooks and ML.NET Pipelines," *ACM SIGMOD Rec.*, vol. 51, no. 2, pp. 30–37, 2022.

[29] "Faceswap." [Online]. Available: https://faceswap.dev/

[30] D. E. Rzig, F. Hassan, C. Bansal and N. Nagappan, "Characterizing the usage of CI tools in ML projects," in *Proc. Int'l Symposium on ESEM*, 2022.

[31] D. G. Widder, D. Nafus, L. Dabbish and J. Herbsleb, "Limits and Possibilities for 'Ethical AI' in Open Source: A Study of Deepfakes," in *Proc. Conf. on Fairness, Accountability, and Transparency*, 2022, pp. 2035–2046.

[32] B. Zhang et al., "Architecture Decisions in AI-based Systems Development: An Empirical Study," in *Proc. Int'l Conf. on Software Analysis, Evolution and Reengineering*, 2022, pp. 616–626.

[33] T. D. Lai et al., "Comparative analysis of real bugs in open-source Machine Learning projects," *arXiv [cs.SE]*, 2022.

[34] D. OBrien et al., "23 shades of self-admitted technical debt: an empirical study on machine learning software," in *Proc. 30th ACM Joint European ESEC/FSE*, 2022, pp. 734–746.

[35] D. Gonzalez et al., "The State of the ML-universe: 10 Years of Artificial Intelligence & Machine Learning Software Development on GitHub," in *Proc. 17th Int'l Conf. on MSR*, 2020, pp. 431–442.

[36] H. Villamizar, M. Kalinowski and H. Lopes, "Towards Perspective-Based Specification of Machine Learning-Enabled Systems," in *Proc. 48th Euromicro Conf. on SEAA*, 2022, pp. 112–115.

[37] J. Siebert et al., "Towards Guidelines for Assessing Qualities of Machine Learning Systems," in *Proc. 13th Int'l Conf. Quality of Information and Communications Technology*, 2020, pp. 17–31.

[38] D. Benyon, *Designing Interactive Systems: A Comprehensive Guide to HCI and Interaction Design*. Addison Wesley, 2010.

[39] J. Robert and A. Lesage, "The Handbook of Human-Machine Interaction : A Human-Centered Design Approach," in *Designing and Evaluating User Experience*, 2017, pp. 321–332.

[40] K. Ulrich, *Product Design and Development*. McGraw-hill, 2016.

[41] "What Is a Product in Product Management?" [Online]. Available: https://airfocus.com/glossary/what-is-a-product-in-product-management/

[42] N. Nahar et al., "A Meta-Summary of Challenges in Building Products with ML Components – Collecting Experiences from 4758+ Practitioners," in *Proc. of 2nd Int'l CAIN*, 2023.

[43] A. Serban and and J. Visser, "Adapting Software Architectures to Machine Learning Challenges," in *Proc. Int'l Conf. on SANER*, 2022, pp. 152–163.

[44] S. Mäkinen, H. Skogström, E. Laaksonen and T. Mikkonen, "Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help?," in *Proc. 1st Workshop on AI Engineering - Software Engineering for AI*, 2021, pp. 109–112.

[45] G. A. Lewis, S. Bellomo and I. Ozkaya, "Characterizing and Detecting Mismatch in Machine-Learning-Enabled Systems," in *Proc. 1st WAIN*, 2021, pp. 133–140.

[46] S. Amershi et al., "Software Engineering for Machine Learning: A Case Study," in *Proc. 41st ICSE-SEIP*, 2019, pp. 291–300.

[47] J. Siegmund, N. Siegmund, and S. Apel, "Views on Internal and External Validity in Empirical Software Engineering," in *Proc. 37th Int'l Conf. on Software Engineering*, 2015, pp. 9–19.

[48] A. Bhat et al., "Aspirations and Practice of ML Model Documentation: Moving the Needle with Nudging and Traceability," in *Proc. of CHI Conf. on Human Factors in Computing Systems*, 2023, pp. 1–17.

[49] L. Quaranta, F. Calefato, and F. Lanubile, "Eliciting Best Practices for Collaboration with Computational Notebooks," *Proc. ACM Hum.-Comput. Interact.*, vol. 6, no. CSCW1, pp. 1–41, 2022.

[50] N. Polyzotis et al., "Data validation for machine learning," in *Proc. Machine Learning and Systems*, 2019, pp. 334–347.

[51] N. Hynes, D. Sculley and M. Terry, "The data linter: Lightweight, automated sanity checking for ml data sets," *NIPS MLSys Workshop*, vol. 1, no. 5, 2017.

[52] A. Ratner et al., "Snorkel: Rapid Training Data Creation with Weak Supervision," *Proc. VLDB Endow.*, vol. 11, no. 3, pp. 269–282, 2017.

[53] C. Ré, F. Niu, P. Gudipati and C. Srisuwananukorn, "Overton: A data system for monitoring and improving machine-learned products," *arXiv [cs.LG]*, 2019.

[54] A. Head et al., "Managing messes in computational notebooks," in *Proc. of CHI Conf. on Human Factors in Computing Systems*, 2019.

[55] S. Amershi et al., "ModelTracker: Redesigning Performance Analysis Tools for Machine Learning," in *Proc. of 33rd Conf. on Human Factors in Computing Systems*, 2015, pp. 337–346.

[56] R. K. E. Bellamy et al., "AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias," *arXiv [cs.AI]*, 2018.

[57] J. Wexler et al., "The What-If Tool: Interactive Probing of Machine Learning Models," *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 1, pp. 56–65, 2020.

[58] B. van Oort et al., "The Prevalence of Code Smells in Machine Learning projects," in *Proc. 1st WAIN*, 2021, pp. 1–8.

[59] Y. Tang et al., "An Empirical Study of Refactorings and Technical Debt in Machine Learning Systems," in *Proc. 43rd ICSE*, 2021, pp. 238–250.

[60] R. Widyasari et al., "NICHE: A Curated Dataset of Engineered Machine Learning Projects in Python," *arXiv [cs.SE]*, 2023.

[61] C. Wan et al., "Automated Testing of Software that Uses Machine Learning APIs," in *Proc. 44th ICSE*, 2022, pp. 212–224.

[62] M. Nehra, "Top 10 Programming Languages for Desktop Apps in 2022." [Online]. Available: https://decipherzone.com/blog-detail/desktop-app-programming-languages

[63] Amjo, "What language are most commonly used for web development." [Online]. Available: https://www.dotnetlanguages.net/web-languages-

what-language-are-most-commonly-used-for-web-development/
[64] Jigsaw, "Perspective API: Using Machine Learning to Reduce Toxicity Online." [Online]. Available: https://www.perspectiveapi.com/
[65] Nahar, N. 2024. Supplementary Documents: The Product Beyond the Model. OSF. [Online]. Available: https://osf.io/gqyex/
[66] D. Piorkowski et al., "How AI Developers Overcome Communication Challenges in a Multidisciplinary Team: A Case Study," *Proc. of the ACM on HCI*, vol. 5, no. CSCW1, pp. 1–25, 2021.
[67] F. Martinez-Plumed et al., "CRISP-DM twenty years later: From data mining processes to data science trajectories," *IEEE Trans. Knowl. Data Eng.*, pp. 1–1, 2020.
[68] E. Breck et al., "The ML test score: A rubric for ML production readiness and technical debt reduction," in *Proc. Int'l Conf. on Big Data*, 2017, pp. 1123–1132.
[69] S. Shankar, R. Garcia, J. M. Hellerstein and A.G. Parameswaran, "Operationalizing Machine Learning: An Interview Study," *arXiv [cs.SE]*, 2022.
[70] B. Rakova, J. Yang, H. Cramer and R. Chowdhury, "Where Responsible AI meets Reality: Practitioner Perspectives on Enablers for shifting Organizational Practices," *Proc. of the ACM on HCI*, vol. 5, no. CSCW1, pp. 1–24, 2020.
[71] S. Rismani and N. Rostamzadeh, "From plane crashes to algorithmic harm: applicability of safety engineering frameworks for responsible ML," in *Proc. CHI Conf. on Human Factors in Computing Systems*, 2023, pp. 1–18.
[72] K. Hammarberg, M. Kirkman, and S. de Lacey, "Qualitative research methods: when to use them and how to judge them," *Hum. Reprod.*, vol. 31, no. 3, pp. 498–501, 2016.
[73] C. Marshall and G. B. Rossman, *Designing Qualitative Research*. SAGE Publications, 2014.
[74] B. Flyvbjerg, "Five Misunderstandings About Case-Study Research," *Qual. Inq.*, vol. 12, no. 2, pp. 219–245, 2006.
[75] S. Harsh, "Purposeful Sampling in Qualitative Research Synthesis," *Qualitative Research Journal*, vol. 11, no. 2, pp. 63–75, 2011.
[76] D. Spencer, *Card Sorting: Designing Usable Categories*. 2009.
[77] E. Kalliamvakou et al., "The Code-Centric Collaboration Perspective: Evidence from Github," *Tech. Report, Uni. of Victoria*, p. 17, 2014.
[78] G. Gousios et al., "Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective," in *Proc. 37th Int'l Conf. on Software Engineering*, 2015, pp. 358–368.
[79] B. Vasilescu et al., "Gender and Tenure Diversity in GitHub Teams," in *Proc. 33rd Conf. on CHI*, 2015, pp. 3789–3798.
[80] K. Yamashita et al., "Revisiting the applicability of the pareto principle to core development teams in open source software projects," in *Proc. 14th IWPSE*, 2015, pp. 46–55.
[81] M. Goeminne and T. Mens, "Evidence for the pareto principle in open source software activity," in *Joint Proc. of the 1st Int'l Workshop on MDSM and 5th Int'l Workshop on SQM*, 2011, pp. 74–82.
[82] T. Zimmermann, S. Diehl, and A. Zeller, "How history justifies system architecture (or not)," in *Proc. 6th Int'l WPSE*, 2003, pp. 73–83.
[83] H. Gall, K. Hajek and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proc. ICSM*, 1998, pp. 190–198.
[84] M. Ferreira et al., "Algorithms for estimating truck factors: a comparative study," *Softw. Qual. J.*, vol. 27, no. 4, pp. 1583–1617, 2019.
[85] G. Avelino et al., "A novel approach for estimating Truck Factors," in *Proc. 24th Int'l Conf. on Program Comprehension*, 2016, pp. 1–10.
[86] A. Serban et al, "Adoption and Effects of Software Engineering Best Practices in Machine Learning," in *Proc. 14th Int'l Symposium on Empirical Software Engineering and Measurement*, 2020, pp. 1–12.
[87] M. L. Libes, "Here's What The Average Tech Startup Looks Like." [Online]. Available: https://www.businessinsider.com/heres-what-the-average-tech-startup-looks-like-2011-12
[88] M. Kim et al., "Data Scientists in Software Teams: State of the Art and Challenges," *IEEE Trans. Software Eng.*, vol. 44, no. 11, pp. 1024–1038, 2018.
[89] A. Hopkins and S. Booth, "Machine Learning Practices Outside Big Tech: How Resource Constraints Challenge Responsible Development," in *Proc. AAAI/ACM Conf. on AI, Ethics, and Society*, 2021, pp. 134–145.
[90] H. Foidl, M. Felderer, and R. Ramler, "Data smells: categories, causes and consequences, and detection of suspicious data in AI-based systems," in *Proc. 1st Int'l CAIN*, 2022, pp. 229–239.
[91] C. Yang et al., "Data Leakage in Notebooks: Static Detection and Better Processes," in *Proc. 37th Int'l Conf. on ASE*, 2023, pp. 1–12.
[92] Z. Buçinca et al., "AHA!: Facilitating AI Impact Assessment by Generating Examples of Harms," *arXiv [cs.HC]*, 2023.
[93] T. Aho et al., "Demystifying Data Science Projects: A Look on the People and Process of Data Science Today," in *Proc 21st Int'l Conf. on Product-Focused Software Process Improvement*, 2020, pp. 153–167.
[94] B. Nushi et al., "On human intellect and machine failures: troubleshooting integrative machine learning systems," in *Proc. 31st AAAI Conf. on Artificial Intelligence*, 2017, pp. 1017–1025.
[95] S. Apel, C. Kästner and E. Kang, "Feature Interactions on Steroids: On the Composition of ML Models," *IEEE Softw.*, vol. 39, no. 3, pp. 120–124, 2022.
[96] H. Yokoyama, "Machine Learning System Architectural Pattern for Improving Operational Stability," in *Proc Int'l Conf. on Software Architecture Companion*, 2019, pp. 267–274.
[97] C. Kästner, *Machine Learning in Production: From Models to Products*. 2022.
[98] V. Lakshmanan, S. Robinson, and M. Munn, *Machine Learning Design Patterns*. O'Reilly Media, Inc., 2020.
[99] H. Washizaki et al., "Machine learning architecture and design patterns," *IEEE Softw.*, vol. 8, 2020.
[100] G. Hulten, *Building Intelligent Systems: A Guide to Machine Learning Engineering*. Apress, 2018.
[101] K. O'Leary and M. Uchida, "Common problems with creating machine learning pipelines from existing code," *MLSys*, 2020.
[102] J. Chang and C. Custis, "Understanding Implementation Challenges in Machine Learning Documentation," in *Proc. Equity and Access in Algorithms, Mechanisms, and Optimization*, 2022, pp. 1–8.
[103] F. Königstorfer and S. Thalmann, "AI Documentation: A path to accountability," *Journal of Responsible Tech.*, vol. 11, p. 100043, 2022.
[104] D. Piorkowski et al., "Towards evaluating and eliciting high-quality documentation for intelligent systems," *arXiv [cs.SE]*, 2020.
[105] M. Arnold et al., "FactSheets: Increasing trust in AI services through supplier's declarations of conformity," *IBM J. Res. Dev.*, vol. 63, no. 4/5, pp. 6:1–6:13, 2019.
[106] M. Mitchell et al., "Model Cards for Model Reporting," in *Proc. Conf. on Fairness, Accountability, and Transparency*, 2019, pp. 220–229.
[107] K. L. Boyd, "Datasheets for Datasets help ML Engineers Notice and Understand Ethical Issues in Training Data," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, no. CSCW2, pp. 1–27, 2021.
[108] W. Ma, C. Yang and C. Kästner, "(Why) Is My Prompt Getting Worse? Rethinking Regression Testing for Evolving LLM APIs," *arXiv [cs.SE]*, 2023.
[109] L. Chen et al., "FrugalML: How to use ML prediction APIs more accurately and cheaply," *Adv. Neural Inf. Process. Syst.*, 2020.
[110] A. K. Paul and M. Schaefer, "Safeguards for the use of artificial intelligence and machine learning in global health," *Bull. World Health Organ.*, vol. 98, no. 4, pp. 282–284, 2020.
[111] H. Abdelkader et al., "ML-On-Rails: Safeguarding Machine Learning Models in Software Systems A Case Study," *arXiv [cs.SE]*, 2024.
[112] A. Cummaudo et al., "Beware the evolving 'intelligent' web service! an integration architecture tactic to guard AI-first components," in *Proc. 28th ACM Joint Meeting on ESEC/FSE*, 2020, pp. 269–280.
[113] M. Vössing et al., "Designing Transparency for Effective Human-AI Collaboration," *Inf. Syst. Front.*, vol. 24, no. 3, pp. 877–895, 2022.
[114] M. Barnett et al., "Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets," in *Proc. 37th Int'l Conf. on Software Engineering*, 2015, pp. 134–144.
[115] A. Brown and G. Wilson, *The Architecture of Open Source Applications, Volume II*. Lulu.com, 2012.
[116] A. Bodor et al., "MLOps: Overview of Current State and Future Directions," in *Proc. Int'l Conf. on SCA*, 2022, pp. 156–165.
[117] S. Li et al., "Testing machine learning systems in industry: an empirical study," in *Proc. of 44th ICSE:SEIP*, 2022, pp. 263–272.
[118] N. Sambasivan et al., "'Everyone wants to do the model work, not the data work': Data Cascades in High-Stakes AI," in *Proc. CHI Conf. on Human Factors in Computing Systems*, 2021.
[119] A. Barrak, E. E. Eghan, and B. Adams, "On the Co-evolution of ML Pipelines and Source Code - Empirical Study of DVC Projects," in *Proc. Int'l Conf. on SANER*, 2021, pp. 422–433.
[120] R. Ashmore, R. Calinescu, and C. Paterson, "Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges," *ACM Comput. Surv.*, vol. 54, no. 5, pp. 1–39, 2021.
[121] D. Kang et al., "Model assertions for monitoring and improving ML models," *MLSys*, vol. abs/2003.01668, 2020.