From Release to Adoption: Challenges in Reusing Pre-trained AI Models for Downstream Developers

Peerachai Banyongrakkul
The University of Melbourne
Melbourne, Australia
pbanyongrakk@student.unimelb.edu.au

Mansooreh Zahedi
The University of Melbourne
Melbourne, Australia
mansooreh.zahedi@unimelb.edu.au

Patanamon Thongtanunam
The University of Melbourne
Melbourne, Australia
patanamon.t@unimelb.edu.au

Christoph Treude
Singapore Management University
Singapore
ctreude@smu.edu.sg

Haoyu Gao
The University of Melbourne
Melbourne, Australia
haoyug1@student.unimelb.edu.au

***Abstract*—Pre-trained models (PTMs) have gained widespread popularity and achieved remarkable success across various fields, driven by their groundbreaking performance and easy accessibility through hosting providers. However, the challenges faced by downstream developers in reusing PTMs in software systems are less explored. To bridge this knowledge gap, we qualitatively created and analyzed a dataset of 840 PTM-related issue reports from 31 OSS GitHub projects. We systematically developed a comprehensive taxonomy of PTM-related challenges that developers face in downstream projects. Our study identifies seven key categories of challenges that downstream developers face in reusing PTMs, such as model usage, model performance, and output quality. We also compared our findings with existing taxonomies. Additionally, we conducted a resolution time analysis and, based on statistical tests, found that PTM-related issues take significantly longer to be resolved than issues unrelated to PTMs, with significant variation across challenge categories. We discuss the implications of our findings for practitioners and possibilities for future research.**

***Index Terms*—pre-trained model, software reuse, taxonomy, open-source software, mining software repositories, qualitative analysis**

I. Introduction

In recent years, AI-based software systems are rapidly spreading in our society, fueled by advancements in Deep Learning (DL) [1]. The development of AI-based software systems differs greatly from traditional software, affecting all stages from requirements to delivery. This is because these systems present greater complexity, involving new stakeholders (e.g., AI developers) with unique concerns (e.g., model accuracy, data quality) [2, 3]. To speed up DL development and reduce costs, pre-trained models (PTMs) are increasingly being used and integrated into systems [4, 5]. PTMs are deep learning models trained on large datasets and widely reused across diverse domains, such as GPT-4 [6].

However, reusing PTMs within software systems introduces new forms of complexity beyond traditional software dependencies [7]. PTMs are often treated as opaque components, with minimal documentation [3], unclear behavior [8], lack of standardized versioning [9], and significant computational demands [10]. As a result, developers who are not involved in model training or architecture design often encounter difficulties in using models for their purposes [11]–[13]. We refer to these developers as downstream developers—that is, practitioners who reuse PTMs in their own software (i.e., downstream software) without directly contributing to the model's development. These developers differ from model creators or ML researchers; they primarily work in software engineering roles and adapt PTMs to fit within their downstream software environments and constraints.

Prior studies have focused on various challenges [5, 12]–[14] and bugs [11, 15] associated with PTMs, but not from the perspective of downstream developers. Davis et al. [5] present a vision paper on PTM reuse challenges without empirical validation, while Jiang et al. [14] and Taraghi et al. [12] focus on the Hugging Face ecosystem, emphasizing limitations within upstream communities. Similarly, Tan et al. [13] analyze PTM-related issues from Stack Overflow discussions, which are often disconnected from real-world development contexts. Meanwhile, Chakraborty et al. [15] and Pan et al. [11] examine bugs within the development of PTMs, rather than challenges in downstream projects. Despite these valuable contributions, little is known about the software development challenges faced by downstream software projects that reuse PTMs from the developer's perspective. To bridge this gap, our study is the first to empirically investigate the PTM-reuse-specific challenges reported by software practitioners in downstream projects on GitHub, where PTMs are reused in actual development environments.

We curated a set of downstream software projects on GitHub that employ PTMs. To investigate the challenges associated with PTM reuse, we systematically extracted issue reports from these repositories. In total, we collected 8,373 issue reports across 31 repositories. Through manual annotation, we identified 840 issue reports related to PTM reuse and thematically analyzed these issues to understand the challenges. Beside constructing a taxonomy of challenges, we also analyzed and compared the resolution times of PTM and

non-PTM issues. In addition, we compared and discussed our results with existing relevant taxonomies and related work.

Our analysis indicates that downstream developers face challenges in projects reusing PTMs across seven key categories. These include issues related to model usage, performance, software environment, documentation, hardware constraints, resource configuration, and functionality enhancements. Notably, most of these challenges are not mentioned in previous work, highlighting the novelty of our taxonomy. Our taxonomy uncovers critical gaps in several aspects, especially in model functionality enhancement and resource configuration. Furthermore, our findings show that PTM issues take significantly longer to be fixed than non-PTM issues, particularly in performance optimization and functionality enhancements. The replication package is available at Zenodo [16].

The key contributions of our research are as follows: 1) a fine-grained taxonomy of challenges when reusing PTMs in downstream software projects, 2) an analysis of resolution times of PTM-related vs. non-PTM-related challenges, 3) a comparative analysis of our results with existing taxonomies and related work, and 4) a replication package with a dataset of 8,373 manually annotated issues from 31 OSS projects reusing PTMs and our qualitative analysis to facilitate future research.

II. Background and Related Work

A. Pre-Trained Model Reuse

Pre-trained models (PTMs) are deep learning models trained on large datasets and made available for reuse across various applications. PTMs have demonstrated significant success across diverse domains, including software engineering [17]. Popular examples of PTMs include BERT [18], GPT-4 [6], Deepseek [19], and Stable Diffusion [20]. Similar to software package reuse, PTMs are widely adopted in software projects and are distributed via model registries like Hugging Face, TensorFlow Hub, PyTorch Hub, and the ONNX Model Zoo. Hugging Face stands out as the largest and most diverse repository [14, 21]. Despite the benefits, reusing PTMs presents its own set of challenges. The black-box nature of these models often leads to software issues, as downstream users may lack a deep understanding of how the model processes data and produces outputs [11, 12], as well as the infrastructure and resources required to run them [22].

Therefore, several recent studies have explored these challenges. The work of Davis et al. [5] envisioned that challenges of PTM reuse would be categorized into three paradigms, including conceptual, adaptation, and deployment reuse, and emphasized standardization and portability concerns. Moreover, Jiang et al. [14] and Taraghi et al. [12] examined PTM reuse in the Hugging Face ecosystem, identifying key challenges such as missing attributes, performance discrepancies, training pipelines, and memory constraints. Tan et al. [13] studied PTM-related discussions on Stack Overflow, revealing challenges in model life cycle management, fine-tuning, and output interpretation. Prior studies also examined bugs related to reusing and the development of PTMs [11, 15]. Chakraborty et al. [15] analyzed Stack Overflow posts on BERT, identifying common bugs about fairness issues, parameter misconfigurations, and version incompatibilities. Pan et al. [11] investigated bugs by mining GitHub repositories for PTM development, finding memory-related and initialization bugs to be prevalent due to PTMs' large size, along with API misuse and retraining errors. The black-box nature of PTMs further exacerbates the difficulties of debugging.

While these studies provide valuable insights into PTM reuse, they primarily focus on challenges from the perspectives of platform users, along with forum discussions [12]–[15], or model repositories [11], capturing high-level or conceptual challenges. Little is known about the challenges faced by downstream software projects that integrate PTMs from a developer's perspective. Our study fills this gap by focusing on GitHub issue reports from downstream software projects reusing PTMs, which provide an empirical understanding of PTM-related challenges faced by software engineers in real-world development environments.

B. Mining GitHub Issues

Recent empirical studies have analyzed GitHub issues to better understand DL frameworks. Makkouk et al. [23] examined performance bugs in DL frameworks (e.g., TensorFlow, PyTorch), noting their complexity and extended resolution time. Long et al. [24] found that many performance and accuracy issues are unclassified or unrelated to actual bugs, with only half leading to direct fixes. Yang et al. [25] proposed a taxonomy of DL framework bugs, identifying model-building bugs as a major challenge requiring specialized fixes. Chen et al. [26] further investigated bugs across four popular DL frameworks by offering a detailed analysis of root causes and symptoms, and providing actionable guidelines to improve bug detection and debugging practices. In addition, some researchers have focused on issues in downstream AI-based systems [10, 27]–[30]. Humbatova et al. [27] developed a taxonomy of challenges in conventional DL systems by analyzing GitHub and Stack Overflow, identifying training-related issues, model-specific faults, and API problems. Yang et al. [28] analyzed open-source AI research projects, highlighting runtime errors and unclear instructions. Zhang et al. [29] examined architecture decisions in AI systems, identifying six linguistic patterns. More recently, Idowu et al. [30] focused on development stages and evolution in ML-related Python projects. Lastly, Lai et al. [10] adapted Humbatova et al.'s taxonomy [27] to analyze ML-applied software projects, comparing ML and non-ML issues in terms of fix duration and size. They found that ML issues take longer to be resolved but show no significant difference in fix size.

These studies provide valuable insights into issue management in DL frameworks and downstream AI-based projects, but they primarily focus on challenges encountered when building traditional ML/DL models from scratch using AI frameworks. They overlook the unique challenges of PTM reuse in downstream projects, where models are integrated, adapted, and maintained, rather than developed from the ground up. This gap highlights the need for an empirical study

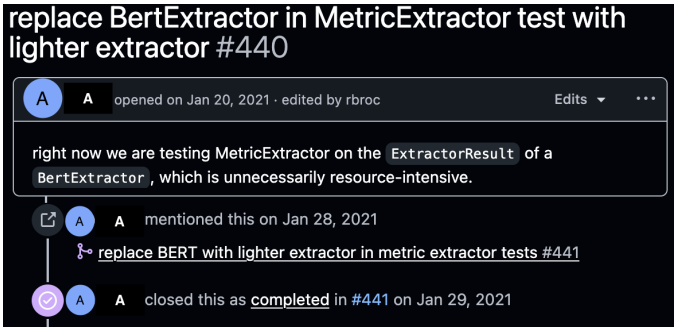


Fig. 1: A motivating example illustrating a PTM-related challenge found in downstream software projects.

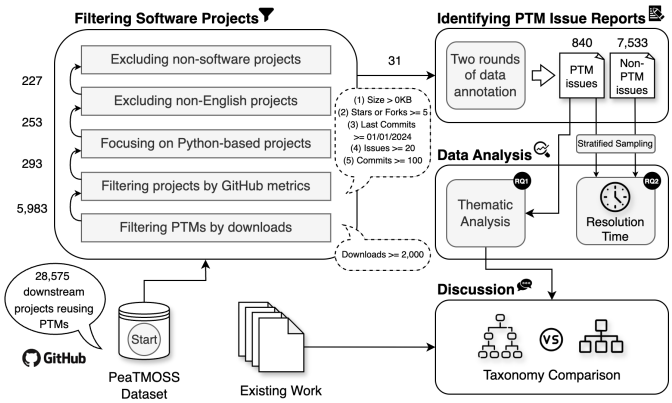


Fig. 2: Overview of our study approach.

focusing on PTM-related issues within real-world software projects reusing PTMs.

C. Motivating Example

In this study, we focus on the challenges downstream developers face when reusing PTMs in real-world software projects. To illustrate these challenges, we present a case from Pliers[1], a software project for developing a Python package for automated multimodal extraction of features from different types of media, such as text, image, audio, and video. In this project, the developers integrated BERT into their feature extraction pipeline to analyze textual data. In the course of development, while BERT offers powerful feature extraction, its high computational cost introduce unnecessary computational overhead, slowing down the pipeline. To address this, a developer created *Pliers #440 [31]* to express the need to replace BERT with a lighter model (i.e., VADER), as shown in Figure 1. This issue reflects a decision-making challenge in PTM integration where developers must balance model capability and resource constraints while developing software. Another example is *Pliers #318 [32]*, where a developer proposed integrating spaCy's PTMs to support their software functionalities. This issue exemplifies the demand for supporting additional models within existing software infrastructures. Integrating new PTMs necessitates careful consideration of compatibility, dependency management, and the potential need for architectural adjustments to accommodate expanded functionalities. These types of challenge do not fit within existing taxonomies, which mainly focus on training-related issues, software environment incompatibilities, model initialization difficulties, and model API usage problems. This demonstrates that PTM reuse in downstream software projects is not only about selecting and applying models, but also involves managing resource constraints (*Pliers #440 [31]*), ensuring compatibility (*Pliers #318 [32]*), and handling versioning issues (*imaginAIry #326 [33]*).

III. Study Design

In this section, we present our research methodology and the study phases, as can be seen in Figure 2.

[1]https://github.com/PsychoinformaticsLab/pliers

A. Research Questions

RQ1: *What PTM-related challenges do downstream developers encounter in software systems?*

Reuse of PTMs comes with its own challenges, spanning both technical limitations and engineering practices [5]. This research question focuses on identifying and categorizing these challenges to better understand the nature of difficulties downstream developers face in their software systems that reuse PTMs.

RQ2: *How does the resolution time of PTM-related issues compare to non-PTM-related issues in software systems?*

Examining the resolution time of PTM-related issues is essential to quantify their impact. Building on RQ1, this research question examines whether PTM-related issues take longer to be resolved than non-PTM issues, inspired by [10]. Investigating these differences helps determine if PTM-specific adaptations of software engineering best practices are needed. This question also explores whether different categories of PTM-related challenges influence resolution time.

B. Data Collection and Pre-processing

For this study, we utilized the PeaTMOSS dataset [7], which contains 28,575 GitHub projects that reuse 281,638 PTMs from Hugging Face and PyTorch Hub. The data was collected as of July 10, 2023. However, it is possible that these GitHub projects may not be software projects (e.g., PTM repositories). Since our goal is to understand the PTM-related issues in the downstream software projects that reuse PTMs, we conduct the following steps to curate the dataset.

Filtering Software Projects: To ensure the quality and relevance of data, we filtered the dataset in the following steps: Firstly, aiming to investigate the challenges of commonly-used PTMs, we included PTMs with more than 2,000 downloads as suggested by previous studies [7, 21, 34, 35], reducing the dataset to 5,938 projects and 197,882 issues. Secondly, we applied GitHub-based metrics, as seen in Figure 2, to ensure the selection of active, well-maintained, and relevant projects. These include size, popularity, activity, data availability, content filtering (excluding tutorials, homework, coding challenges, or resource collections via keyword-based

filtering from [36]), issues, and commits. These criteria are adapted from best practices in software repository analysis [11, 36]–[40]. Notably, the thresholds in both steps were chosen based on elbow points, where the curve levels off when plotting each metric against the number of remaining projects. This resulted in 293 projects and 116,658 issues. Then, we included only projects written in Python, as it is the most popular programming language for AI-related projects [37, 41] and the predominant language in the dataset, yielding 253 projects and 101,596 issues. After that, we manually inspected and excluded projects where most content was in a non-English language, such as *Bark-Voice-Cloning*[2], resulting in 227 projects and 93,061 issues.

Finally, we excluded non-software projects through manual validation. Specifically, this involved inspecting repository descriptions, READMEs, and structures to include only projects reusing PTMs as components for their core functionality, targeting non-AI developers or engineers, and not PTM repositories themselves. Firstly, we removed non-technical projects (e.g., research papers, books). Moreover, repositories dedicated to the development of PTMs (e.g., *ConvLab-3*[3]), or repositories of AI tools specifically designed for AI developers, like model benchmarking, training, or deployment (e.g., *chitra*[4]). To ensure rigor and consensus, manual inspections were performed collaboratively by two authors. Specifically, the first author randomly selected a sample of 50 projects and shared them with another author. Two authors independently annotated them, achieving a Cohen's kappa of 0.65, indicating substantial agreement [42]. Disagreements were resolved collaboratively before applying the criteria to the remaining records by the first author. The disagreements often arose in borderline cases. One example of a grey-area case is *FormFyxer*[5], which was initially debated as it is developed by a research group. After discussion, we agreed to include it, as it demonstrates real-world software functionality and clear evidence of software engineering practice. Ultimately, 37 projects with 8,671 issues remained, encompassing a variety of software. For example, AI-powered applications (e.g., image and video generation[6]), domain-specific AI-based tools (e.g., medical concept annotation[7]), and frameworks and libraries (e.g., a Python package for feature extraction[1]).

TABLE I: Statistics of the selected downstream repositories

Metric \ Statistics	Min	Max	Mean	SD
Stars	11	26,230	3,219.52	6,377.21
Forks	1	2,571	384.00	732.17
Commits	190	6,426.00	1,046.57	1,236.87
Issues	28	1,871	201.00	383.25
PTM-Related Issues	1	239	27.10	48.98

[2]https://github.com/KevinWang676/Bark-Voice-Cloning
[3]https://github.com/ConvLab/ConvLab-3
[4]https://github.com/aniketmaurya/chitra
[5]https://github.com/SuffolkLITLab/FormFyxer
[6]https://github.com/brycedrennan/imaginAIry
[7]https://github.com/CogStack/MedCAT

Identifying PTM-Related Issue Reports: With the filtered software projects, we conducted two rounds of data annotation to identify PTM-related issue reports. The first author randomly selected 100 issues and shared them with another author. Both, with more than four years of ML/DL experience, independently labeled each issue as 'PTM' or 'Non-PTM' based on the title, description, and discussions. The first round yielded a Cohen's kappa of 0.60 (moderate agreement) [42]. After resolving conflicts, we refined our inclusion/exclusion criteria. In the second round, another 100 reports were annotated using the revised criteria, improving Cohen's kappa to 0.82 (strong agreement) [42]. The remaining disagreements were resolved, further refining the criteria. Our inclusion criteria explicitly encompass diverse issue types (e.g., bugs, enhancement requests, feature discussions, and general questions), as long as they reflect challenges that developers face related to PTM reuse. These PTM-related issues were identified within three main components of AI-based software: the model itself, the data pipeline, and the training processes, inspired by [2, 5]. Additionally, we included issues that discuss AI practices and supporting elements such as software environment, system infrastructure, and documentation. Exclusion criteria filter out non-PTM issues, including generic bugs, non-model components, research content, non-English issues, and issues with insufficient detail. After reaching consensus, the first author applied the finalized criteria to annotate the remaining issues. Projects without PTM-related issues were removed, resulting in 840 PTM-related issue reports and 7,533 non-PTM issues across 31 projects. Table I presents a descriptive summary of key statistics for the dataset. This dataset [16] provides a diverse foundation for analyzing PTM reuse in different software development environments.

C. Data Analysis

In this section we elaborate on the process of analyzing data to address our research questions.

Thematic Analysis (RQ1): After completion of data preparation, we qualitatively analyzed 840 PTM-related issue reports using thematic analysis [43, 44] to establish our fine-grained taxonomy of PTM-related challenges. All analyses were performed using NVivo[8]. Initially, to become familiar with the data, the first author randomly selected a pilot sample of 50 issue reports for manual inspection with another author. During this phase, we extracted keypoints of challenges that developers encountered, labeled them with preliminary descriptive tags (i.e., codes), and categorized them accordingly. Through this step, we established common understanding to ensure consistency in our data analysis. For instance, a single issue report could contain one or multiple distinct challenges. We also excluded challenges that were too vague, overly broad, or lacked sufficient information for meaningful analysis. These initial codes were then reviewed and discussed by all authors to align the coding scheme and ensure consistency before proceeding with the full dataset.

[8]https://lumivero.com/products/nvivo

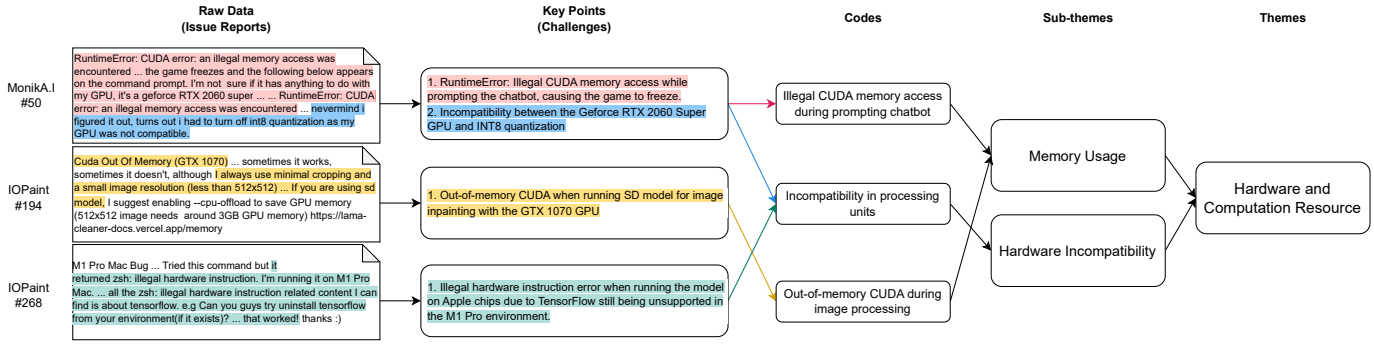


Fig. 3: An example of the data analysis process for developing the taxonomy of PTM-related challenges.

Throughout the full analysis, we employ a purely bottom-up methodology, iteratively comparing new codes with existing ones to refine and adjust categories based on emerging themes, as exemplified in Figure 3. The first author was primarily responsible for this process, which involved the following key steps. Firstly, we systematically reviewed issue reports by analyzing their titles, descriptions, and discussions, with additional reference to related issues and pull requests where applicable. Secondly, we extracted challenges and assigned leaf-level codes to them. Finally, emerging codes were iteratively grouped into higher-level themes (i.e., sub-themes and themes) to capture overarching patterns. To maintain quality and consistency, the assignment and categorization of the codes were reviewed and revised every 50 issue reports. All the other authors closely monitored the entire process through weekly meetings. This iterative approach allowed for continuous refinement, where codes were adjusted, merged, split, or removed as needed. In cases of uncertainty or disagreement, discussions were held to reach a final consensus, with conflicts being systematically resolved in regular meetings.

Resolution Time Analysis (RQ2): Our objective in this analysis is to determine whether PTM-related issues require more time to resolve than non-PTM issues. Resolution time is defined as the number of days from creation of an issue to its closure. If an issue is closed on the same day it is opened, its resolution time is zero. To enable a fair comparison, we first constructed two balanced groups of closed issue reports—PTM-related and non-PTM—by employing stratified sampling [45]. For each project, we randomly selected an equal number of closed non-PTM issues to match the number of closed PTM-related issues. If a project had fewer available non-PTM issues, we proportionally reduced the number of PTM issues to maintain balance. Projects where one group had no closed issues were excluded. This process resulted in a dataset of 802 issue reports—401 PTM-related and 401 non-PTM-related—across 27 projects.

To further mitigate potential confounding effects, particularly those stemming from differences in issue types, we used a fine-tuned seBERT model [46] to classify issues into three categories: bug, question, and enhancement. These are the predefined issue types supported by the model, and the classification was used solely to provide a high-level overview of issue type distribution and validate the balance between the PTM and non-PTM groups. To strengthen the validity of this classification, we conducted a broader manual validation. Using a statistically representative sample of 260 issues (130 PTM and 130 non-PTM) with 95% confidence level and 5% margin of error, we found that 249 cases (95.77%) matched the manual classification, confirming the reliability of the seBERT-based issue type tagging. A Chi-square independence test [47] confirmed that the distribution of issue types did not significantly differ between the groups (p-value = 0.2), supporting the comparability of the two sets.

For the statistical comparison of resolution times, we employed the one-sided Mann-Whitney U Test [48], a non-parametric test, with significance level of 0.05 to determine whether PTM-related issues take significantly longer to resolve than non-PTM issues. Additionally, we conducted a Kruskal-Wallis H-test [49], a non-parametric test, to determine if the medians of PTM-related challenge themes differ, using the same significance level as the previous test. We also computed the average resolution time for PTM-related issues across different taxonomy themes and sub-themes. This allowed us to analyze which challenge categories required the longest resolution time.

IV. Taxonomy of Challenges (RQ1)

In this section, we present our final taxonomy of PTM-related challenges. The taxonomy is structured into three hierarchical levels [16]. Figure 4 shows the seven top-level themes (gray boxes) including: (1) Model Usage, (2) Model Performance and Output Quality, (3) Software Environment, (4) Developer Information and Documentation, (5) Hardware and Computation Resource, (6) Model Resource and Configuration, and (7) Model Functionality Enhancement Needs. These themes are further divided into 32 sub-themes (white boxes). In the figure, each challenge group is labeled in the format (number of challenges, number of issue reports, number of codes). For instance, the group Model IO Handling is annotated as (93, 92, 7), indicating that this sub-theme encompasses 93 individual challenge instances drawn from 92 issue reports, and these were abstracted into 7 unique codes.

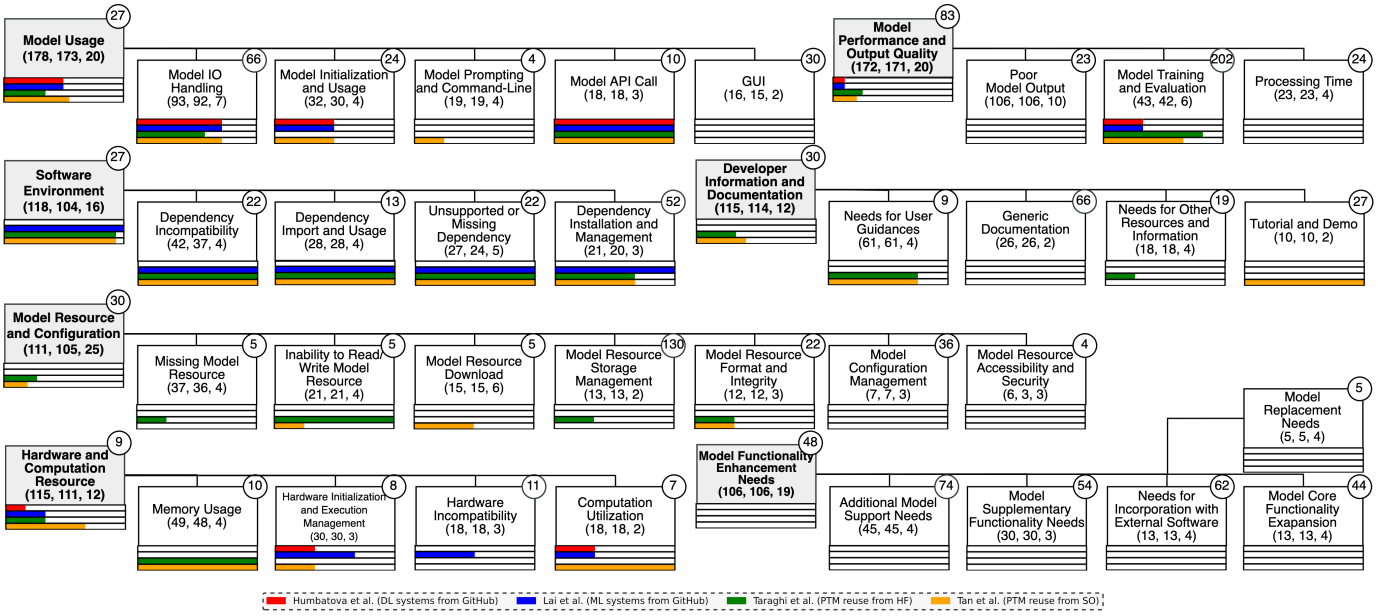


Fig. 4: The top two levels of our taxonomy of PTM-related challenges in downstream software projects, where each challenge group is labeled as (# challenges, # issue reports, # codes). The colored bars represent the overlap with existing taxonomies [10, 12, 13, 27], and empty bars indicate new categories. The small circles at the top-right indicate the average resolution time.

A. Model Usage (MU)

This theme is the most frequent in our taxonomy, accounting for 178 challenges. It covers issues related to interacting with PTMs. We elaborate on the sub-themes of this theme below:

Model I/O Handling (93 challenges): Developers usually face difficulties in handling model input and output, including passing parameters as tensor inputs through a model. One of the most common issues involves mismatches in types or shapes of data or tensors, such as *Threestudio #260 [50]*. Another frequent challenge is saving model outputs, exemplified in *SimSwap #133 [51]*. Model I/O issues also arise from input/output formatting, such as image padding errors and JSON conversion failures. Encoding and tokenization inconsistencies further contribute to these challenges, including embedding mismatches and handling unknown tokens. Developers also struggle with refining model outputs, underscoring the need for systematic post-processing, particularly for string matching and misspelling corrections.

Model Initialization and Usage (32 challenges): Developers face compatibility issues, missing attributes, and structural mismatches when initializing PTMs. Most issues arise during model training and functionality usage from the user side, such as *SimSwap #416 [52]*, indicating a missing model attribute that prevents initialization. Structural mismatches, like hidden layer discrepancies, like *SimSwap #251 [51]*, also cause errors.

Model Prompting and Command-Line (19 challenges): Some software enables user interaction with models via prompts and CLI, but developers often encounter challenges in handling command-line parameters, such as missing model options (*IOPaint #90 [53]*). Formatting issues also arise, including excessive technical details in prompt templates and handling malformed inputs. Additionally, developers receive feature requests for custom model support, as seen in *imaginAIry #255 [54]*.

Model API Call (18 challenges): Developers often encounter issues when interacting with model providers like OpenAI and Hugging Face, particularly with API calls for remote models. Authorization errors (*IOPaint #89 [55]*) are the most common. API quota limits also cause disruptions, as seen in *babyagi #32 [56]*. Also, server connection failures, including bad gateway errors and connection resets, make it difficult to maintain stable communication with model providers.

GUI (16 challenges): Developers face challenges with GUI malfunctions and usability improvements. Issues like failed model selection, missing input fields, and incorrect defaults cause unexpected behavior (*NLP-Suite #1010 [57]*). Although GUI issues may seem generic, in the context of PTM reuse, they often directly affect model functionality and user control over PTM behavior. For instance, in *NLP-Suite #876 [58]*, a GUI failure prevented users from modifying memory-related configurations, which are essential for executing PTMs. This indicates a strong coupling between GUI elements and PTM-specific resource demands. Beyond fixing functionality, they also aim to enhance usability specific to PTM workflows, such as intuitive controls and direct model checkpoint selection.

B. Model Performance and Output Quality (PQ)

Our analysis shows that ensuring high-quality model performance is one of the most frequent challenges (172) that developers face when reusing PTMs. In the following, we elaborate on the sub-themes of the category.

Poor Model Output (106 challenges): Developers often report missing, inconsistent, or distorted model outputs, especially in image and video tasks like inpainting. Issues include blank images, artifacts, and glitches (*IOPaint #292 [59]*). Similar challenges occur in text processing, where outputs are misformatted, exemplified in *AutoPR #65 [60]*.

Model Training and Evaluation (43 challenges): This subtheme covers challenges in model training and fine-tuning, from dataset preparation to evaluation. Developers face issues with invalid or insufficient data, such as incompatible formats and misaligned labels (*Zamba #234 [61]*). Training misconfigurations, including unregistered learning rate schedulers and missing validation splits, disrupt workflows. Fine-tuning struggles arise from optimization constraints, lack of built-in support, and inefficient multi-GPU utilization. Evaluation challenges include monitoring gaps, poor visualization tools and inadequate validation metrics, exemplified in *OpenBB #4136 [62]*. Slow training speeds and caching inefficiencies further limit scalability, highlighting the demand for more efficient pipelines. There are also needs for retraining a model to improve accuracy, as shown in *Zamba #54 [63]*.

Processing Time (23 challenges): Processing speed is another bottleneck in PTM-powered applications, with developers reporting slow or inconsistent inference times, especially in image generation. For example, *ImaginAIry #113 [64]* highlights significant performance discrepancies across software versions. Some developers found ineffective processing time in a specific hardware setting, as seen in *IOPaint #270 [65]*.

C. Software Environment (SE)

Developers also frequently encounter difficulties when dealing with software environment around PTMs. This theme includes 118 challenges and is divided into four sub-themes:

Dependency Incompatibility (42 challenges): Dependency incompatibility is the most prevalent challenges in this subtheme, with developers frequently encountering four main types of conflicts: library-to-library, Python version mismatches, library-to-model, and library-to-OS incompatibilities. For instance, strict versioning conflicts are evident in *IOPaint #315 [66]*. Similarly, Python version incompatibilities can cause build failures, as reported in *MedCAT #34 [67]*.

Dependency Import and Usage (28 challenges): Developers frequently face challenges when importing and using dependencies, particularly PTM-related libraries, like transformers, as seen in *imaginAIry #352 [68]*. They also failed to import other AI-related libraries, such as pytorch and stanza. Beyond import failures, incorrect usage also leads to runtime errors, such as *Scattertext #16 [69]*.

Unsupported or Missing Dependencies (27 challenges): Missing or unsupported dependencies hinder model execution, especially AI libraries (e.g., sacremoses, transformers) and system tools, exemplified in *StoryToolkitAI #40 [70]*. Hardware acceleration libraries, like CUDA and cuDNN also cause compatibility issues. For example, *Threestudio #128 [71]*, a RuntimeError occurs due to an unsupported OpenGL version.

Dependency Installation and Management (21 challenges): Managing dependencies through requirements files is essential for maintaining version integrity and preventing inconsistencies. However, developers usually encounter challenges related to package updates, renaming, and installation failures. For instance, package renaming by providers can cause unexpected disruptions, as seen in *StoryToolkitAI #47 [72]*. Installation failures are another common issue, particularly when required package versions are unavailable, as reported in *NLP-Suite #47 [73]*.

D. Developer Information and Documentation (DI)

Developers often struggle with insufficient user guidance, missing or inaccurate documentation, and a lack of essential resources, which hinder usability and accessibility to their model functionality. This theme consists of 115 challenges and is divided into four sub-themes:

Needs for User Guidance (61 challenges): This sub-theme covers the how-to questions that developers face. A popular question is how to use functionality with user's custom model, as seen in *SimSwap #245 [74]*. Similarly, with some software offering the user option for fine-tuning, questions on training and performance optimization arise, such as *SimSwap #363 [75]*. Other queries involve model I/O, resource handling, and optimizing models for specific hardware and environments.

Generic Documentation (26 challenges): Lack of clear and accurate documentation remains a major barrier to developing PTM-based software. Developers often struggle with missing or incomplete guides, as illustrated in *app.enfugue.ai #25 [76]*. Beyond missing resources, inaccurate documentation, including outdated instructions, typos, and inconsistencies, which further complicates the functionality usage for the users.

Needs for Other Resources and Information (18 challenges): Beyond standard documentation, developers often experience requests for additional resources and technical clarifications to better understand and extend PTM functionality. Common requests include original model training settings, training scripts, backend implementation details, and lists of supported models. For instance, a user requested access to the training file in *EditAnything #33 [77]*.

Tutorial and Demo (10 challenges): Incomplete or inaccurate tutorials and demos also create significant barriers between PTM-related fucntionalities and users. Many struggle with outdated guides, misleading instructions, and missing content for critical tasks such as model usage, training, and inference, as shown in *MedCAT #152 [78]*.

E. Hardware and Computation Resource (HR)

Efficient hardware utilization is critical for PTM-applied software, yet developers frequently encounter challenges related to GPU and CPU in terms of memory constraints and ineffective computational resource usage. This theme comprises 115 challenges and is divided into four sub-themes:

Memory Usage (49 challenges): PTMs are highly memory-intensive, often exceeding hardware limits and causing execution failures. Out-of-memory errors (*IOPaint #3 [79]*) and

insufficient memory allocation dominate this sub-theme, making model execution infeasible under constrained hardware. Developers also struggle with illegal memory access and the lack of memory optimization, such as *MonikA.I #50 [80].*

Hardware Initialization and Execution Management (30 challenges): Managing hardware resources poses significant challenges, with developers facing CUDA failures, execution provider issues, and multiprocessing inefficiencies. Missing CUDA devices (*IOPaint #104 [81]*) are also common, along with ONNXRuntime CUDA loading failures. Additionally, multiprocessing issues, such as improperly released semaphore objects used for process synchronization, contribute to execution failures.

Hardware Incompatibility (18 challenges): Hardware mismatches, such as unsupported processing units and suboptimal tensor allocations, create bottlenecks in PTM execution. For example, *imaginAIry #300 [82]* reflects a device mismatch issue, where tensors were placed on different devices (MPS and CPU). Some models fail to leverage hardware accelerators effectively due to compatibility issues, such as Pytorch MPS not supporting the specific model (*IOPaint #286 [83]*).

Computation Utilization (18 challenges): Even when hardware is available, balancing CPU-GPU workloads and optimizing resource allocation remain important as the previous one. Developers often report inefficient GPU utilization and poor CPU-GPU assignment strategies, such as *SimSwap #192 [84].*

F. Model Resource and Configuration (RC)

We also observed challenges related to the management of model resources and configurations. In this theme, 111 challenges are covered and categorized into seven sub-themes:

Missing Model Resource (37 challenges): Developers often encounter missing model files and model checkpoint files, required for model usage, inference, or resuming training, such as *MedCAT #19 [85].* In addition, missing configuration files and text embedding files further complicate model functionality execution, forcing developers to reconfigure or re-download the necessary resources.

Inability to Read-Write Model Resource (21 challenges): Encoding, serialization, and decompression issues frequently obstruct model reading and writing processes. For example, developers found a Unicode encoding error, as seen in *SimSwap #288 [86].*

Model Resource Download (15 challenges): Downloading PTM resources is another challenge, with developers reporting HTTP connection failures and authorization restrictions (*imaginAIry #12 [87]*). Developers also face inefficient downloading mechanisms, as exemplified in *Nomic #164 [88].*

Model Resource Storage Management (13 challenges): Developers face challenges in locating, organizing, and storing model resource files, including inconsistent directory structures, storage path conflicts, and difficulties with dynamic imports, as shown in *Marie-ai #20 [89].* Large model files also exceed storage limits, causing checkpoint failures and caching issues, especially in constrained environments like Colab.

Model Resource Format and Integrity (12 challenges): Developers face compatibility issues when dealing with model formats, structural inconsistencies, and integrity verification failures, as seen in *SimSwap #173 [90]* and *SimSwap #9 [91].*

Model Configuration Management (7 challenges): Handling model configurations remains a challenge, with developers encountering incorrect default configuration parameter settings and lack of flexibility in modifying configurations. Moreover, developers highlight the need for centralized configuration management, as exemplified in *MedCAT #72 [92].*

Model Resource Accessibility and Security (6 challenges): Developers also receive reports from users about denied access to model resources, such as vocabulary files. Additionally, developers encounter security issues. For example, in *MedCAT #65 [93]*, a developer reported that they suffered spam attacks downloading their model files, causing a service disruption. A similar case is *BabyAGI #15 [94]*, mismanagement of API keys, probably creating security risks and unexpected costs.

G. Model Functionality Enhancement Needs (FN)

Despite the bugs found during PTM integration, developers usually face enhancement requests. This category has 106 challenges and is divided into five sub-themes:

Additional Model Support Needs (45 challenges): Developers receive numerous requests for supporting additional models, especially for image-related tasks like generation, embedding, inpainting, and 3D reconstruction. For example, *imaginAIry #337 [95]* shows a demand for models with superior performance in specific scenarios. Requests also extend to other domains, such as text (e.g., SPECTER and BERT), audio (e.g., YAMNet), and video (e.g., Whisper).

Model Supplementary Functionality Needs (30 challenges): Developers also face the demands for developing the supplementary features to the existing core functionality for several purposes. Enhancing user experience is a key priority, with users seeking better interfaces, usability refinements, and more accessible configuration options, such as *SimSwap #18 [96].* Additional requests focus on optimizing model performance, output quality, and operational efficiency.

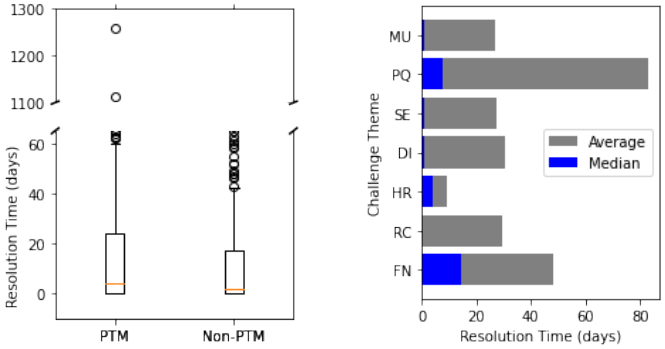
Model Core Functionality Expansion Needs (13 challenges): Aside from supplementary functionalities, they also receive requests to expand the model capabilities to cover additional core features, particularly in image and text-related tasks. These include image-to-text, image-to-3D, outpainting, refinement, and negative prompt-based generation, primarily driven by user demands, such as *Threestudio #65 [97].* Additional requests include audio and video features, such as non-English speech and multi-language transcription from video.

Needs for Incorporation with External Software (13 challenges): PTM-applied software often receives requests for third-party integrations to enhance usability and functionality. For example, *babyagi #105 [98]* suggests integrating SingularGPT for UI automation using natural language. Other requests focus on performance optimization, cost reduction, and cross-platform compatibility.

Model Replacement Needs (5 challenges): Apart from adding and expanding model capabilities, developers also receive requests to replace existing models, as illustrated in Figure 1 and discussed in the motivation section. These replacements are often driven by the need to improve resource efficiency. Other motivations include performance optimization, cost reduction, and version upgrades, as exemplified in *imaginAIry #326 [33].*

V. Resolution Time (RQ2)

Figure 5a presents box plots that compare resolution times for PTM and non-PTM issues. As observed, PTM-related issues take longer to resolve, averaging 40 days compared to 26 days for non-PTM issues. In terms of minimum and maximum resolution times, both groups include issues that were resolved on the same day (0 days). However, PTM-related issues exhibit a much longer maximum resolution time of 1,260 days, compared to 507 days for non-PTM issues. In addition, the Mann-Whitney U Test confirms that PTM-related issues take significantly longer to resolve than non-PTM issues, with a p-value of 0.047.



(a) PTM vs. Non-PTM. (b) Fix Duration by Theme.

Fig. 5: Resolution time analysis of PTM-related issues.

Figure 5b presents the average and median resolution times across top-level challenge themes. A Kruskal-Wallis H-test reveals a statistically significant difference in resolution time across different challenge categories ($p < 0.001$). This indicates that certain challenge categories take disproportionately longer to resolve than others. *Model Performance and Output Quality (PQ)* has the highest average resolution time (79 days). The longest sub-theme is Model Training and Evaluation, averaging 202 days. Within this sub-theme, Model Training Needs, which includes requirements for model retraining and fine-tuning to improve accuracy, was the most prolonged issue. Note that only the top-level themes are shown in Figure 5b.

The findings also show that *Model Functionality Enhancement Needs (FN)* has the second-highest average resolution time, at 48 days. We found that issues under this theme often involve enhancements rather than traditional bug fixes. These issues require developers to assess multiple factors before deciding to implement them, such as implementation complexity, availability of existing models or functionalities, project scope limitations, accuracy concerns, potential risks of misuse, and development costs. Notably, we observed multiple cases where feature requests were rejected outright, not due to technical infeasibility, but rather because they failed to meet the above considerations, as exemplified in *babyagi #105 [98]*, *imaginAIry #116 [99]*, and *imaginAIry #132 [100].*

VI. Discussion

A. Overview of Challenges in PTM Reuse

From Figure 4, out of a total of 915 challenges, we observed that the most frequently discussed PTM-related challenges in downstream software projects are *Model Usage* and *Model Performance and Output Quality*. These findings highlight a critical need for improved handling of model I/O (i.e., the highest number of challenges within *Model Usage*), and addressing issues related to poor-quality model outputs (i.e., the most prevalent among all sub-themes). Conversely, *Model Functionality Enhancement Needs* and *Model Resource and Configuration* cover fewer challenges. Despite this difference, the overall range between the most and least discussed challenges remains relatively balanced, indicating a uniform distribution of difficulties encountered by developers. Furthermore, the high frequency of documentation-related issues highlights the need for clearer and accurate user guidance, which remains insufficient, as reflected in the numerous challenges under *Developer Information and Documentation*. Lastly, even challenges with lower frequencies, such as *Model Resource Accessibility and Security*, should not be overlooked, as neglecting them could potentially lead to significant negative impacts.

B. Taxonomy Comparison

We compare our taxonomy (RQ1) with existing taxonomies of the four most relevant and recent studies [10, 12, 13, 27] to discuss unique and additional challenges found in downstream software projects reusing PTM. For a precise comparison, we create a mapping between the codes in our taxonomy and those in the existing taxonomies at the lowest level [16]. We analyze their codes based on their explanations, sample issues, and the replication package. Where conceptual overlap existed, we aligned our codes accordingly, with each of their codes mapping to none, one, or multiple codes in our taxonomy, depending on the degree of overlap. The color bars below each box in Figure 4 show the percentage of overlapping codes where white bars indicate new challenges. Overall, we found that most of the codes identified in our study were distinct from existing taxonomies, especially in *Model Functionality Enhancement Needs*, *Model Resource and Configuration*, and *Model Performance and Output Quality*.

Comparing with the challenges in traditional DL/ML-based GitHub projects by Humbatova et al. [27] (red bar) and Lai et al. [10] (blue bar) in Figure 4, their taxonomies extensively cover *Model Usage* issues, particularly in model I/O handling, model initialization, and API interactions as they mainly focus on developing traditional models from scratch. Although Lai et al. [10] further address *Software Environment* challenges, they overlook *Model Resource and Configuration*, which is critical

in PTM reuse (e.g., managing model files, checkpoints, and embeddings), leading to storage challenges. The challenges of *Developer Information and Documentation*, *Model Performance and Output Quality* issues (e.g., fine-tuning difficulties) and prompting challenges in *Model Usage* were not found in traditional AI-based projects. This highlights the increased complexity of PTM-applied software, where both development and usage require extensive technical support and resources.

For the taxonomies of PTM reuse mined from Stack Overflow discussions by Tan et al. [13] (yellow bar) and model repositories on Hugging Face by Taraghi et al. [12] (green bar) in Figure 4, the complexities of PTM reuse in real-world downstream projects were not captured. A remarkable omission is *Model Functionality Enhancement Needs*. Unlike other model users, downstream software projects demand additional model support, model replacement, core and supplementary functionality expansion, and third-party software integration. As discussed in our resolution time analysis, these enhancement requests require developers to evaluate feasibility, complexity, and software constraints before implementation, highlighting the increased effort required when reusing PTMs. Many challenges were also not captured in prior taxonomies [12, 13]. For example, GUI-related challenges as PTM-powered applications frequently incorporate GUIs to enhance usability. Although both taxonomies include *Developer Information and Documentation* challenges, the unclear documentation when using PTMs in specific applications was not captured. Other aspects such as poor model output, processing time bottlenecks, and computational utilization issues also were not captured in the prior taxonomies, highlighting the unique challenges of PTM reuse which are far beyond model selection and adaptation.

C. PTM-related Resolution Time

For the discussion on resolution time, the findings from RQ2 underscore that PTM issues take longer to be resolved than non-PTM issues. A possible reason behind this is that many PTM issues go beyond simple bug identification and fixing. Instead, they involve model performance optimization, which was found to have the highest average fix time. This process is typically more complex and iterative. Debugging these issues often requires retraining models or modifying data pipelines, tasks that demand considerable time. *Model Functionality Enhancement Needs* could be another contributing factor, as we observed that issues under this theme often depend on multiple aspects, requiring developers to invest additional effort to resolve them. Interestingly, we observed that while *Model Usage* is the most prevalent challenge developers face, issues under this category tend to be resolved more quickly than most other themes, with the exception of *Hardware and Computation Resource*, which has the shortest fix time.

VII. Implications

A. Implications for Practitioners

Understanding the challenges can help downstream software developers anticipate pitfalls, refine development practices, and implement proactive strategies. We recommend the followings: ① **Enhancing Model Functionality Performance & Customizability**—Based on RQ1, integrating monitoring tools could help streamline PTM training and improve model performance and output quality. Supporting lightweight model-based functionality (e.g., [101]–[104]) and multi-processing options would enable users to run models with limited resources. Additionally, developers should implement support for custom or user-specified models (e.g., from Hugging Face or local storage) via command-line prompting. ② **Improved Documentation & Usability Support**—As indicated by our RQ1 findings, user guidance issues are one of the dominant challenges. Developers should provide comprehensive and versioned documentation, especially clear usage instructions (e.g., prompting, CLI, GUI, hardware requirements) and ensure alignment with evolving dependencies. Additionally, we suggest that developers employ automated tools [39, 105] to simplify technical documentation, thereby enhancing user accessibility. ③ **Streamlining Resource Management & Configuration**—To address the challenges in *Model Resource and Configuration*, particularly the sub-theme with the longest fix duration—Model Resource Management, standardized configuration formats (e.g., YAML [106]) may help centralize settings in a single file, making management easier. Implementing model resource caching can minimize redundant downloads, improving efficiency for end users. ④ **Strategic Resource Allocation for PTM Issue Resolution**—From RQ2, PTM-related issues take longer to resolve, particularly in model training issues and requests for model functionality enhancements. Developers can benefit from allocating extra effort, prioritizing tasks by expected resolution time [10], and documenting common failures. Moreover, clear acceptance criteria [107] may also help manage enhancement requests.

B. Implications for Researchers

Our systematic analysis of PTM reuse challenges can help researchers identify current gaps, inspire new research directions, and develop novel solutions. Based on our findings, we propose the following research directions: ① **Automated PTM Issue Resolution & Effort Estimation**—The results of RQ2 indicate that PTM-related issues vary in complexity and resolution time. We suggest researchers to explore automated tools for prioritizing [108], estimating effort [109], and predicting resolution duration [110] specifically for PTM issues, aiding developers in better resource allocation. ② **Enhancing Feature Request Evaluation & PTM Evolution**—Based on the findings of RQ2, many requests from *Model Functionality Enhancement Needs* are declined due to implementation complexity, project scope limitations, or security concerns. This raises key research questions: How can software engineering practices better support PTM evolution? What factors influence the adoption and replacement of PTMs in OSS projects, and how do developers decide which models to support or replace? [7] What trade-offs exist between expanding PTM functionality and maintaining software stability? ③ **Lightweight PTM Development &**

Hardware Efficiency—Given the resource-intensive nature of PTMs, highlighted by the prominence of memory usage issues in RQ1, we suggest that researchers explore techniques to optimize model efficiency, including lightweight architectures [111], quantization and hardware-aware model design [112] to improve accessibility across various computing environments. ④ **Addressing PTM Compatibility & Environment Challenges**—Incompatibilities between PTM-related dependencies hinder smooth integration. Future research could focus on automated tools for conflict detection and resolution, alternative library suggestions, and improved compatibility mechanisms, as emphasized in existing work [13, 112].

VIII. Threats to Validity

A. Internal Validity

Filtering software projects and PTM-related issue reports have not been previously studied in the literature. To reduce bias, we define clear inclusion/exclusion criteria and employ two annotators for manual classification with inter-annotator agreement measurement. In the first annotation phase, which involved filtering software projects, the agreement score was 0.65. Although this may appear moderate, it is generally considered a substantial level of agreement based on the guidelines by Landis et al. [113], and aligns with accepted standards in software engineering research [114]. In the second phase, which focused on identifying PTM-related issues, the initial agreement was slightly lower at 0.60, but improved significantly to 0.82 in the second round after a reconciliation discussion. While manual challenge categorization for RQ1 involves some subjectivity [115], we follow a rigorous approach [43, 44], including a pilot study for annotator familiarization and iterative refinement through weekly discussions. Additionally, resolution time in RQ2 may be misleading as an issue report can contain multiple challenges, probably skewing the distribution. However, such cases are limited, and we address this by using non-parametric tests to assess median differences, reducing sensitivity to outliers.

B. External Validity

Our base dataset, PeaTMOSS [7], is externally sourced which might introduce unknown biases. However, we mitigate potential bias through multiple rigorous filtering steps, including manual annotation with agreement checks. Additionally, in comparative analysis, some prior studies exhibit inconsistencies between the taxonomies described in their papers and the resources provided in their replication packages, while others lack detailed coverage of software environment issues. To address this, we thoroughly examine all available issue report examples. Furthermore, to enhance transparency, we include the comparative evidence in our replication package [16].

C. Construct Validity

Our study aims to identify challenges specifically related to the reuse of PTMs in downstream software projects. However, there is a risk that a few identified sub-themes may appear generic to general software systems (e.g., GUI). To mitigate this threat, we focus our analysis on how these challenges manifest themselves uniquely in the context of PTM reuse. In RQ2, resolution time may not always reflect actual fix effort, as compounding factors such as issue type, developer workload, and limited interaction with the reporter may influence it [40]. To mitigate this threat, we controlled the distribution for two significant factors—project and issue type—by employing stratified random sampling when selecting non-PTM issues to match the PTM group. Both groups were drawn from the same annotated dataset, and the matching process ensured a balanced distribution across projects and issue types. To validate the reliability of the issue type classification by seBERT used for stratification, we also performed a broader manual validation of a representative sample to indicate that the use of seBERT introduces minimal risk of systematic bias. While this reduces sampling bias and improves comparability, we acknowledge that residual confounding factors and sensitivity to sample variation may still influence results. We also accounted for extreme outliers by using non-parametric tests, ensuring our analysis is less sensitive to anomalies.

IX. Conclusion

This paper presents a comprehensive and systematic analysis of the challenges derived from PTM-related issue reports found in downstream OSS GitHub projects. We manually conducted rigorous data filtering to create a curated dataset of 840 PTM-related issue reports across 31 OSS projects reusing PTMs. Our data analysis reveals seven main themes of challenges: Model Usage, Model Performance and Quality, Software Environment, Developer Information and Documentation, Hardware and Computation Resource, Model Resource and Configuration, and Model Functionality Enhancement Needs. We also found that PTM-related issues take significantly longer to resolve than non-PTM issues.

Based on the taxonomy comparisons, we found that the majority of our identified challenges remain unaddressed in existing work, emphasizing the novelty of our findings. Existing studies primarily focus on Model Usage and Software Environment issues, while our taxonomy introduces new areas in Model Functionality Enhancement Needs, Model Resource and Configuration, and Model Performance and Output Quality. We also provided practical recommendations for downstream developers and researchers to improve the practices in this field, highlighting the need for more robust tooling, better documentation, systematic strategies in software development with PTMs, and future research directions.

References

[1] S. Martíinez-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," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 2, apr 2022.

[2] H. Muccini and K. Vaidhyanathan, "Software architecture for ML-based Systems: What exists and what lies ahead," in *Proceedings of the IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN 2021)*. IEEE, 2021, pp. 121–128.

[3] H. Gao, M. Zahedi, C. Treude, S. Rosenstock, and M. Cheong, "Documenting ethical considerations in open source ai models," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2024)*. New York, NY, USA: Association for Computing Machinery, 2024, p. 177–188.

[4] X. Han and et al., "Pre-trained models: Past, present and future," *AI Open*, vol. 2, pp. 225–250, 2021.

[5] J. C. Davis, P. Jajal, W. Jiang, T. R. Schorlemmer, N. Synovic, and G. K. Thiruvathukal, "Reusing Deep Learning Models: Challenges and Directions in Software Engineering," in *Proceedings of the 2023 IEEE John Vincent Atanasoff Symposium on Modern Computing (JVA 2023)*, 2023, pp. 17–30.

[6] OpenAI, "GPT-4 Technical Report," vol. 4, pp. 1–100, 2023.

[7] 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 *Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024)*, vol. 1, 2024.

[8] J. Burrell, "How the machine 'thinks': Understanding opacity in machine learning algorithms," *Big Data & Society*, vol. 3, no. 1, p. 2053951715622512, 2016.

[9] A. Ajibode, A. A. Bangash, F. R. Cogo, B. Adams, and A. E. Hassan, "Towards semantic versioning of open pre-trained language model releases on hugging face," *Empirical Software Engineering*, vol. 30, no. 3, 2025.

[10] T. D. Lai, A. Simmons, S. Barnett, J. G. Schneider, and R. Vasa, "Comparative analysis of real issues in open-source machine learning projects," *Empirical Software Engineering*, vol. 29, no. 3, 2024.

[11] R. Pan, S. Biswas, M. Chakraborty, B. D. Cruz, and H. Rajan, "An Empirical Study on the Bugs Found while Reusing Pre-trained Natural Language Processing Models," 2022.

[12] M. Taraghi, G. Dorcelus, A. Foundjem, F. Tambon, and F. Khomh, "Deep Learning Model Reuse in the HuggingFace Community: Challenges, Benefit and Trends," in *Proceedings of the 31st IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2024)*. IEEE, mar 2024, pp. 512–523.

[13] X. Tan, T. Li, R. Chen, F. Liu, and L. Zhang, "Challenges of Using Pre-trained Models: the Practitioners' Perspective," in *Proceedings of the 31st IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2024)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2024, pp. 67–78.

[14] W. Jiang, N. Synovic, M. Hyatt, T. R. Schorlemmer, R. Sethi, Y. H. 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 *Proceedings of the 45th International Conference on Software Engineering (ICSE 2023)*, 2023, pp. 2463–2475.

[15] M. Chakraborty, "Does reusing pre-trained NLP model propagate bugs?" in *Proceedings of the 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*, 2021, pp. 1686–1688.

[16] P. Banyongrakkul, M. Zahedi, P. Thongtanunam, C. Treude, and H. Gao, "A Replication Package of the Paper: "From Release to Adoption: Challenges in Reusing Pre-trained AI Models for Downstream Developers"," mar 2025. [Online]. Available: http://doi.org/10.5281/zenodo.15020621

[17] A. Karmakar and R. Robbes, "What do pre-trained code models know about code?" in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021)*, 2021, pp. 1332–1336.

[18] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019)*, vol. 1, oct 2019, pp. 4171–4186.

[19] DeepSeek-AI, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," vol. 500, pp. 1–22, 2025.

[20] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, " High-Resolution Image Synthesis with Latent Diffusion Models ," in *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR 2022)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2022, pp. 10 674–10 685.

[21] J. Castaño, S. Martínez-Fernández, X. Franch, and J. Bogner, "Exploring the Carbon Footprint of Hugging Face's ML Models: A Repository Mining Study," in *Proceedings of the 17th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2023)*. IEEE, oct 2023, pp. 1–12.

[22] C. Zhou and et al., "A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT," *International Journal of Machine Learning and Cybernetics*, 2023.

[23] T. Makkouk, D. J. Kim, and T. H. Chen, "An Empirical Study on Performance Bugs in Deep Learning Frameworks," in *Proceedings of the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME 2022)*, 2022, pp. 35–46.

[24] G. Long and T. Chen, "On Reporting Performance and Accuracy Bugs for Deep Learning Frameworks: An Exploratory Study from GitHub," in *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering (EASE 2022)*, 2022, pp. 90–99.

[25] Y. Yang, T. He, Z. Xia, and Y. Feng, "A comprehensive empirical study on bug characteristics of deep learning frameworks," *Information and Software Technology*, vol. 151, no. July, p. 107004, 2022.

[26] J. Chen, Y. Liang, Q. Shen, J. Jiang, and S. Li, "Toward Understanding Deep Learning Framework Bugs," *ACM Transactions on Software Engineering and Methodology*, 2023.

[27] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proceedings of the 42nd International Conference on Software Engineering (ICSE 2020)*, 2020, pp. 1110–1121.

[28] Z. Yang, C. Wang, J. Shi, T. Hoang, P. Kochhar, Q. Lu, Z. Xing, and D. Lo, "What Do Users Ask in Open-Source AI Repositories? An Empirical Study of GitHub Issues," in *Proceedings of the 20th IEEE/ACM International Conference on Mining Software Repositories (MSR 2023)*, 2023, pp. 1–13.

[29] B. Zhang, T. Liu, P. Liang, C. Wang, M. Shahin, and J. Yu, "Architecture Decisions in AI-based Systems Development: An Empirical Study," in *Proceedings of the 30th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2023)*, 2023.

[30] S. Idowu, Y. Sens, T. Berger, J. Krueger, and M. Vierhauser, "A Large-Scale Study of ML-Related Python Projects," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing (SAC 2024)*, 2024, pp. 1272–1281.

[31] Pliers #440: http://github.com/PsychoinformaticsLab/pliers/issues/440.

[32] Pliers #318: http://github.com/PsychoinformaticsLab/pliers/issues/318.

[33] ImaginAIry #326: http://github.com/brycedrennan/imaginAIry/issues/326.

[34] F. Pepe, V. Nardone, A. Mastropaolo, G. Canfora, G. Bavota, and M. Di Penta, "How do Hugging Face Models Document Datasets, Bias, and Licenses? An Empirical Study," in *Proceddings of the 32nd IEEE/ACM International Conference on Program Comprehension (ICPC 2024)*, vol. 1, no. iii, 2024.

[35] W. Jiang, N. Synovic, P. Jajal, T. R. Schorlemmer, A. Tewari, B. Pareek, G. K. Thiruvathukal, and J. C. Davis, "PTMTorrent: A Dataset for Mining Open-source Pre-trained Model Packages," in *Proceedings of the 20th IEEE/ACM International Conference on Mining Software Repositories (MSR 2023)*, 2023, pp. 57–61.

[36] T. R. Toma and C. P. Bezemer, "An exploratory study of dataset and model management in open source machine learning applications," in *Proceedings of the 3rd IEEE/ACM International Conference on AI Engineering - Software Engineering for AI (CAIN 2024)*, 2024.

[37] 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 *Proceedings of the 17th IEEE/ACM International Conference on Mining Software Repositories (MSR 2020)*, 2020, pp. 431–442.

[38] D. Obrien, S. Biswas, S. Imtiaz, R. Abdalkareem, E. Shihab, and H. Rajan, "23 Shades of Self-Admitted Technical Debt: an Empirical Study on Machine Learning Software," in *Proceedings of the 30th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)*, 2022, pp. 734–746.

[39] H. Gao, C. Treude, and M. Zahedi, "Evaluating transfer learning for simplifying github readmes," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)*. New York, NY, USA: Association for Computing Machinery, 2023, p. 1548–1560.

[40] P. Banyongrakkul and S. Phoomvuthisarn, "DeepPull: Deep Learning-Based Approach for Predicting Reopening, Decision, and Lifetime of Pull Requests on GitHub Open-Source Projects," in *Communications in*

Computer and Information Science, vol. 2104 CCIS. Springer Nature Switzerland, 2024, pp. 100–123.
[41] 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," in *Proceedings of the 44th International Conference on Software Engineering (ICSE 2022)*, vol. 2022-May, 2022, pp. 2091–2103.
[42] M. L. McHugh, "Interrater reliability: the kappa statistic." *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.
[43] D. S. Cruzes and T. Dybå, "Recommended steps for thematic synthesis in software engineering," in *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement (ESEM 2011)*, no. 7491. IEEE, 2011, pp. 275–284.
[44] V. Braun and V. Clarke, "Using thematic analysis in psychology; In qualittaive research in psychology," *Uwe Bristol*, vol. 3, no. 2, pp. 77–101, 2006.
[45] S. Baltes and P. Ralph, "Sampling in software engineering research: a critical review and guidelines," *Empirical Software Engineering*, vol. 27, no. 4, 2022.
[46] A. Trautsch and S. Herbold, "Predicting issue types with seBERT," in *Proceedings of the 1st International Workshop on Natural Language-Based Software Engineering (NLBSE 20 22)*. New York, NY, USA: Association for Computing Machinery, 2023, pp. 37–39.
[47] M. L. McHugh, "The chi-square test of independence." *Biochemia medica*, vol. 23, no. 2, pp. 143–149, 2013.
[48] H. B. Mann and D. R. Whitney, "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.
[49] W. H. Kruskal and W. A. Wallis, "Use of Ranks in One-Criterion Variance Analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.
[50] Threestudio #260: http://github.com/threestudio-project/threestudio/issues/260.
[51] SimSwap #133: http://github.com/neuralchen/SimSwap/issues/133.
[52] SimSwap #416: http://github.com/neuralchen/SimSwap/issues/416.
[53] IOPaint #90: http://github.com/Sanster/IOPaint/issues/90.
[54] ImaginAIry #255: http://github.com/brycedrennan/imaginAIry/issues/255.
[55] IOPaint #89: http://github.com/Sanster/IOPaint/issues/89.
[56] BabyAGI #32: http://github.com/yoheinakajima/babyagi/issues/32.
[57] NLP-Suite #1010: http://github.com/NLP-Suite/NLP-Suite/issues/1010.
[58] NLP-Suite #876: https://github.com/NLP-Suite/NLP-Suite/issues/876.
[59] IOPaint #292: http://github.com/Sanster/IOPaint/issues/292.
[60] AutoPR #65: http://github.com/irgolic/AutoPR/issues/65.
[61] Zamba #234: http://github.com/drivendataorg/Zamba/issues/234.
[62] OpenBB #4136: http://github.com/OpenBB-finance/OpenBB/issues/4136.
[63] Zamba #54: http://github.com/drivendataorg/Zamba/issues/54.
[64] ImaginAIry #113: http://github.com/brycedrennan/imaginAIry/issues/113.
[65] IOPaint #270: http://github.com/Sanster/IOPaint/issues/270.
[66] IOPaint #315: http://github.com/Sanster/IOPaint/issues/315.
[67] MedCAT #34: http://github.com/CogStack/MedCAT/issues/34.
[68] ImaginAIry #352: http://github.com/brycedrennan/imaginAIry/issues/352.
[69] Scattertext #16: http://github.com/JasonKessler/Scattertext/issues/16.
[70] StoryToolkitAI #40: http://github.com/octimot/StoryToolkitAI/issues/40.
[71] Threestudio #128: http://github.com/threestudio-project/threestudio/issues/128.
[72] StoryToolkitAI #47: http://github.com/octimot/StoryToolkitAI/issues/47.
[73] NLP-Suite #47: http://github.com/NLP-Suite/NLP-Suite/issues/47.
[74] SimSwap #245: http://github.com/neuralchen/SimSwap/issues/245.
[75] SimSwap #363: http://github.com/neuralchen/SimSwap/issues/363.
[76] App.enfugue.ai #25: http://github.com/painebenjamin/app.enfugue.ai/issues/25.
[77] EditAnything #33: http://github.com/sail-sg/EditAnything/issues/33.
[78] MedCAT #152: http://github.com/CogStack/MedCAT/issues/152.
[79] IOPaint #3: http://github.com/Sanster/IOPaint/issues/3.
[80] MonikA.I #50: http://github.com/Rubiksman78/MonikA.I/issues/50.
[81] IOPaint #104: http://github.com/Sanster/IOPaint/issues/104.
[82] ImaginAIry #300: http://github.com/brycedrennan/imaginAIry/issues/300.
[83] IOPaint #286: http://github.com/Sanster/IOPaint/issues/286.
[84] SimSwap #192: http://github.com/neuralchen/SimSwap/issues/192.
[85] MedCAT #19: http://github.com/CogStack/MedCAT/issues/19.
[86] SimSwap #288: http://github.com/neuralchen/SimSwap/issues/288.
[87] ImaginAIry #12: http://github.com/brycedrennan/imaginAIry/issues/12.
[88] Nomic #164: http://github.com/nomic-ai/nomic/issues/164.
[89] Marie-ai #20: http://github.com/marieai/marie-ai/issues/20.
[90] SimSwap #173: http://github.com/neuralchen/SimSwap/issues/173.
[91] SimSwap #9: http://github.com/neuralchen/SimSwap/issues/9.
[92] MedCAT #72: http://github.com/CogStack/MedCAT/issues/72.
[93] MedCAT #65: http://github.com/CogStack/MedCAT/issues/65.
[94] BabyAGI #15: https://github.com/yoheinakajima/babyagi/issues/15.
[95] ImaginAIry #337: http://github.com/brycedrennan/imaginAIry/issues/337.
[96] SimSwap #18: http://github.com/neuralchen/SimSwap/issues/18.
[97] Threestudio #65: http://github.com/threestudio-project/threestudio/issues/65.
[98] BabyAGI #105: http://github.com/yoheinakajima/babyagi/issues/105.
[99] ImaginAIry #116: http://github.com/brycedrennan/imaginAIry/issues/116.
[100] ImaginAIry #132: http://github.com/brycedrennan/imaginAIry/issues/132.
[101] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "Quartznet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020)*, vol. 2020-May, 2020, pp. 6124–6128.
[102] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.0, 2019.
[103] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017.
[104] S. Gandhi, P. von Platen, and A. M. Rush, "Distil-Whisper: Robust Knowledge Distillation via Large-Scale Pseudo Labelling," 2023.
[105] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, "ARENA: An Approach for the Automated Generation of Release Notes," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, pp. 106–127, 2017.
[106] F. Cák and P. Dakić, "Creating Feature Model for YAML Generator in CI/CD Pipelines with React Web Application," in *Proceedings of the 9th International Congress on Information and Communication Technology (ICICT 2024)*, X.-S. Yang, S. Sherratt, N. Dey, and A. Joshi, Eds. Singapore: Springer Nature Singapore, 2024, pp. 529–539.
[107] C. J. Wallace D., "Guide to software Acceptance," vol. 53, no. 9, pp. 1689–1699, 2019.
[108] Y. Bugayenko, A. Bakare, A. Cheverda, M. Farina, A. Kruglov, Y. Plaksin, W. Pedrycz, and G. Succi, "Prioritizing tasks in software development: A systematic literature review," pp. 1–31, 2023.
[109] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637–656, 2019.
[110] P. Banyongrakkul and S. Phoomvuthisarn, "Multi-output learning for predicting evaluation and reopening of github pull requests on open-source projects," in *Proceedings of the 18th International Conference on Software Technologies (ICSOFT 2023)*, INSTICC. SciTePress, 2023, pp. 163–174.
[111] H. Long, W. Bi, and J. Sun, "Lightweight Design and Optimization methods for DCNNs: Progress and Futures," pp. 1–18, 2024.
[112] M. Dilhara, A. Ketkar, and D. Dig, "Understanding Software-2.0: A Study of Machine Learning Library Usage and Evolution," *ACM Transactions on Software Engineering and Methodology*, vol. 30, 2021.
[113] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data." *Biometrics*, vol. 33, no. 1, pp. 159–174, mar 1977.
[114] F. Elberzhager, J. Münch, and V. T. N. Nha, "A systematic mapping study on the combination of static and dynamic quality assurance techniques," *Information and Software Technology*, vol. 54, no. 1, pp. 1–15, 2012.
[115] C. Ratner, "Subjectivity and objectivity in qualitative methodology," *Forum Qualitative Sozialforschung*, vol. 3, no. 3, 2002.