PeaTMOSS: A Dataset and Initial Analysis of Pre-Trained Models in Open-Source Software

Wenxin Jiang
Purdue University
W Lafayette, IN, USA
jiang784@purdue.edu

Jerin Yasmin
Queen's University
Kingston, ON, CA
19jy2@queensu.ca

Jason Jones
Purdue University
W Lafayette, IN, USA
jone2078@purdue.edu

Nicholas Synovic
Loyola University Chicago
Chicago, IL, USA
nsynovic@luc.edu

Jiashen Kuo
Purdue University
W Lafayette, IN, USA
kuo90@purdue.edu

Nathaniel Bielanski
Purdue University
W Lafayette, IN, USA
nbielans@purdue.edu

Yuan Tian
Queen's University
Kingston, ON, CA
y.tian@queensu.ca

George K. Thiruvathukal
Loyola University Chicago
Chicago, IL, USA
gkt@cs.luc.edu

James C. Davis
Purdue University
W Lafayette, IN, USA
davisjam@purdue.edu

Abstract

The development and training of deep learning models have become increasingly costly and complex. Consequently, software engineers are adopting pre-trained models (PTMs) for their downstream applications. The dynamics of the PTM supply chain remain largely unexplored, signaling a clear need for structured datasets that document not only the metadata but also the subsequent applications of these models. Without such data, the MSR community cannot comprehensively understand the impact of PTM adoption and reuse.

This paper presents the *PeaTMOSS* dataset, which comprises metadata for 281,638 PTMs and detailed snapshots for all PTMs with over 50 monthly downloads (14,296 PTMs), along with 28,575 open-source software repositories from GitHub that utilize these models. Additionally, the dataset includes 44,337 mappings from 15,129 downstream GitHub repositories to the 2,530 PTMs they use. To enhance the dataset's comprehensiveness, we developed prompts for a large language model to automatically extract model metadata, including the model's training datasets, parameters, and evaluation metrics. Our analysis of this dataset provides the first summary statistics for the PTM supply chain, showing the trend of PTM development and common shortcomings of PTM package documentation. Our example application reveals inconsistencies in software licenses across PTMs and their dependent projects. PeaTMOSS lays the foundation for future research, offering rich opportunities to investigate the PTM supply chain. We outline mining opportunities on PTMs, their downstream usage, and cross-cutting questions.

MSR 2024, April 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04…$15.00
https://doi.org/10.1145/3643991.3644907

Our artifact is available at https://github.com/PurdueDualityLab/PeaTMOSS-Artifact. Our dataset is available at https://transfer.rcac.purdue.edu/file-manager?origin_id=ff978999-16c2-4b50-ac7a-947ffdc3eb1d&origin_path=%2F.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; **Information extraction**; • **Information systems** → **Database design and models**; • **Software and its engineering** → **Software libraries and repositories**.

Keywords

Datasets, Machine learning, Deep neural networks, Model zoos, Package registries, Open-source, Empirical software engineering

ACM Reference Format:
Wenxin Jiang, Jerin Yasmin, Jason Jones, Nicholas Synovic, Jiashen Kuo, Nathaniel Bielanski, Yuan Tian, George K. Thiruvathukal, and James C. Davis. 2024. 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 '24), April, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3643991.3644907

1 Introduction

Deep Neural Networks (DNNs) have become a common component in software systems over the past decade. Developing and training DNN models is costly, requiring specialized hardware and large datasets [36, 76]. While some software engineers develop DNNs from scratch, others integrate DNNs into software following a typical reuse pattern [26, 57, 70]: (1) pre-trained DNN models (PTMs) are published to registries such as Hugging Face (analogous to traditional package registries such as NPM); and (2) other software depends on these PTMs, accessed by library or web API.

Despite the widespread adoption of PTMs [59, 88], our understanding of the software engineering practices and challenges surrounding PTM reuse remains limited [54]. This understanding is critical for developing more sophisticated tools, mitigating risks,

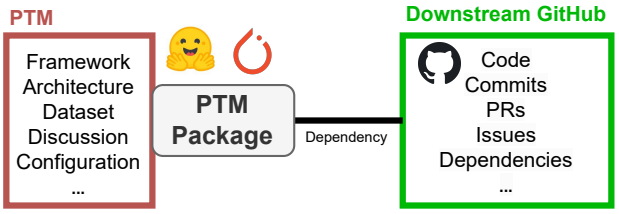

Figure 1: This paper presents the *PeaTMOSS* dataset: Pre-Trained Models in Open-Source Software. *PeaTMOSS* includes data on 281,638 pre-trained models, 28,575 GitHub repositories that use pre-trained models, and 44,337 links between them.

and guiding best practices [9]. Mining Software Repositories techniques could help, but unfortunately current datasets on PTMs lack crucial details, which leaves a gap in knowledge [8, 55]. For instance, they frequently omit comprehensive evaluation metrics, model training conditions, parameters, and standardization in reporting results. This absence of information impedes our ability to perform robust analyses, compare performances meaningfully, or derive a coherent picture of PTMs' impact and usage in software engineering. Recent work highlights the need for a more complete spectrum of metadata is required [54, 55], which should include—but not be limited to—details on model training datasets, versioning, licensing, and the computational requirements for PTM reuse.

To address this gap, the primary contribution of this work is the creation of the `PeaTMOSS` dataset: **P**re-**T**rained **M**odels in **O**pen-**S**ource **S**oftware. *PeaTMOSS* enables mining of PTMs, the software projects that use them, and the interactions between PTMs and downstream use. As illustrated in Figure 1, *PeaTMOSS* contains a snapshot of: (1) 281,638 PTMs, (2) 28,575 open-source software repositories that use PTMs, providing real-world context for how these models are applied, and (3) 44,337 mappings between PTMs and downstream GitHub repositories. Our secondary contribution involves the practical application of large language models (LLMs) to extract PTM metadata, thereby enhancing our dataset (§5). We apply this tool to systematically extract key metadata, including datasets, hyper-parameters, and performance metrics, from unstructured model cards. Li *et al.* called for comprehensive metadata to construct a queryable model zoo, enabling efficient search and comparison of models [67]. By addressing the challenges of unstructured data, we ensure that our model zoo encompasses a wide range of critical information, facilitating more informed and precise queries.

We conduct two demonstrations of the value of this dataset. In §6 we analyze the data distribution in *PeaTMOSS*, show the trends in the growth of PTM development and identify the common shortcomings in PTM package documentation. In §7 we use the mapping created for PTM and GitHub projects to analyze the consistency of software licenses. As future work, the PeaTMOSS dataset offers many opportunities to study and inform our understanding of the PTM supply chain. We propose three distinct directions for analyzing *PeaTMOSS*: (1) analyses focusing on the GitHub data subset, (2) explorations centered on the PTM aspect, and (3) comprehensive studies integrating insights from both GitHub and PTM components. We suggest researchers take advantage of the *PeaTMOSS* dataset and conduct a larger-scale measurement on characterizing the properties of the PTM supply chain. **Our contributions are**:

- We share a dataset named PeaTMOSS which includes 281,638 PTM packages, and 28,575 downstream GitHub repositories.
- We tackled the issue of unstructured attributes by developing a LLM-based tool for metadata extraction, which enhances our dataset by adding structured data in JSON format.
- We provide the first summary statistics of this PTM supply chain, encompassing distributions of PTMs and their downstream repositories across various problem domains. Our analysis also includes trends in model size and the quantity of PTM packages, along with an overview of the proportion of available metadata. We show the proportion of missing data in each PTM metadata category.
- We applied our dataset to assess the compatibility of PTMs with downstream GitHub repositories. Our findings reveal that 0.24% of these licenses are inconsistent, potentially causing community confusion and hindering collaboration.

Significance: *PeaTMOSS* is a comprehensive dataset for PTM in open-source software. It offers an extensive mapping between PTM packages and downstream GitHub repositories, and many queryable metadata. Using *PeaTMOSS*, researchers can study the PTM supply chain and the reuse modes of PTM packages. Engineering tools can be developed for PTM reuse, *e.g.*, for model search and comparison.

Paper outline: This paper is organized as follows: §2 and §3 provide background and related work. In §4, we describe the original version of the *PeaTMOSS* dataset, and §5 details the augmented dataset enriched with our metadata extraction pipeline. Data analysis of *PeaTMOSS* is presented in §6. §7 illustrates a practical application. The paper concludes with an examination of potential threats to validity in §8, followed by a discussion of future work in §9.

2 Background

This section covers PTMs (§2.1) and their reuse (§2.2).

2.1 Pre-Trained Deep Learning Models (PTMs)

The advent of deep learning has precipitated a fundamental shift in computational methodologies, transitioning from the deterministic algorithms characteristic of traditional software to increasingly probabilistic and data-driven paradigms [58]. Deep learning typically operates through neural networks capable of assimilating datasets, thereby enabling them to make predictions or perform complex tasks [62]. A PTM embodies a DNN architecture that has undergone prior training with a specific dataset, incorporating a defined data pipeline, training regime, and learned parameters ("weights"). This pre-training equips the PTM to perform inference or to be adapted for downstream applications [26].

Existing research has explored various methods for reusing deep learning models, such as feature extraction, transfer learning, data generation, and model compression [46, 56]. For instance, DNNs can be pre-trained using large-scale unlabeled molecular databases and then fine-tuned over specific chemical downstream tasks like molecular property prediction [103]. Additionally, models can be

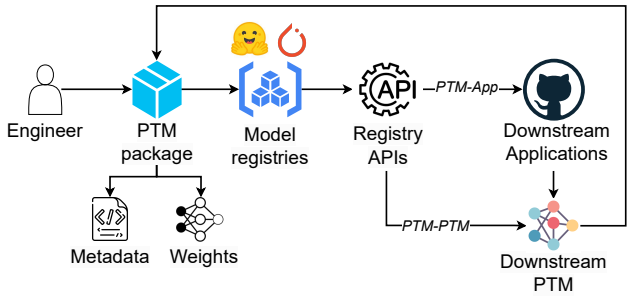

Figure 2: The PTM supply chain. Engineers publish PTM packages to model registries. PTMs are used by applications and other PTMs.

employed to annotate data or to synthesize new datasets through generative approaches [10, 29]. Transfer learning enables models trained on generic datasets to refine their understanding of more detailed, downstream tasks, often resulting in enhanced performance on specialized datasets [112]. Furthermore, models can be optimized for size and efficiency to run on edge devices, a process known as model compression [27].

Thanks to this range of reuse modes, in recent years PTMs have become increasingly popular [22, 59]. The total number of open-source PTM packages has seen a consistent increase on a monthly basis [22]. Table 1 provides a quantitative demonstration of the extensive adoption and rising popularity of PTMs. Previous research indicates that the popularity and adoption rate of Hugging Face's models are comparable to those of other established software package registries, including npm and PyPI [54].

Table 1: Comparison of package counts and download figures for the top 10% of PTMs on Hugging Face. Data for August 2022 is sourced from the PTMTorrent dataset [55]. The August 2023 data is obtained from our dataset. This comparison highlights the growth of PTM usage over a one-year period (*i.e.,* doubling).

Hugging Face Statistics	Aug. 2022	Aug. 2023
# packages of all PTMs	124 K	282 K
# downloads of top 10% PTMs	269 billion	587 billion

2.2 Components of the PTM Supply Chain

Jiang *et al.*. introduced the PTM supply chain concept, encompassing PTM packages, the model registries, the authors of PTMs, and the users [56]. Figure 2 extends their model to include the downstream applications, thus providing a holistic view of the PTM ecosystem. *PeaTMOSS* contains the major elements of this supply chain. This section describes each element in turn.

2.2.1 PTM Packages. PTMs are often shared in *PTM packages*. Per Jiang *et al.* [54], a PTM package is analogous to traditional software packages on platforms like NPM or PyPI [3, 4]. A PTM package has standard elements such as a license, documentation, and usage examples. Analogous to source code, a PTM package describes the model architecture and pre-trained weights. Its metadata indicates the training regime, which includes the dataset(s) involved, how the model's parameters were initialized, and the necessary data pre- and post-processing ("data pipeline"). A PTM package may indicate the model's performance on evaluation metrics.

2.2.2 Model Registries. PTM packages are commonly disseminated via deep learning model registries (also known as model hubs/zoos). Jiang *et al.* define a deep learning model registry as *a collaborative hub where teams share deep learning models* [54]. Prior work shows that there are three kinds of model registries categorized by their contribution types [56]: open (*e.g.,* Hugging Face [33]), gated (*e.g.,* PyTorch Hub [78]), and commercial (*e.g.,* NVIDIA NGC catalog [6]). These platforms enable engineers to directly adopt PTMs or adapt them through fine-tuning for specialized downstream tasks.

2.2.3 Package Dependencies. The various methodologies for PTM reuse establish two distinct types of dependencies within the PTM supply chain. Firstly, there are *PTM-PTM* dependencies, where, for instance, one model might be fine-tuned from another [56]. Secondly, there are *PTM-Application* dependencies, where software projects rely on PTMs for their functionality [59]. These dependencies underscore the interconnected nature of PTM reuse and highlight an aspect of PTM package usage that necessitates further exploration, particularly in how these dependencies impact the broader software engineering landscape.

3 Related Work

This section covers related work on software engineering in PTM reuse (§3.1), importance of queryable PTM metadata (§3.2), and open-source PTM datasets (§3.3).

3.1 Software Engineering in PTM Reuse

Prior work has comprehensively studied the development of deep learning systems from software engineering perspectives [9, 81]. These works more focused on creating and training new DNNs from scratch, which usually requires extensive resources and expertise. However, the reuse process of PTM focused on adapting existing PTMs which is a different process compared to developing a new model [54]. The literature of understanding the reuse of PTMs still presents a notable gap.

Davis *et al.* introduced three paradigms for reusing DNNs: conceptual reuse, adaptation reuse, and deployment reuse [26]. Prior work has characterized conceptual reuse in the form of DNN model reengineering and proposed the challenges in this reuse tpye, including performance debugging, and portability of deep learning operations [52]. In the context of adapting PTMs in the application, there are two main challenges faced by software engineers: (1) technical adoption challenges, and (2) decision-making challenges such as model selection and evaluation [26]. For deployment reuse, Jajal *et al.* characterized failures of deep learning model converters which could compromise model quality [49]

Recent empirical research highlights the popularity of PTM registries among engineers. They appreciate these registries for their well-organized problem domains and user-friendly APIs, which are vital for downstream applications [54, 56, 93]. Studies by Jiang *et al.* and others have identified distinct differences between traditional software package reuse and PTM package reuse. These differences include varied decision-making processes, unique attributes that facilitate reuse, and specific risk factors relevant in PTM contexts. The impact of PTMs on software engineering practices has been a focal point of recent studies [22, 59]. Gong *et al.* have explored

the usage contexts of PTM packages via an exploratory study from model hubs, but there is still a substantial gap in understanding the detailed reuse of these models [42]. Our dataset complements these findings by providing a detailed mapping between PTM packages and downstream GitHub repositories. This enables further, more insightful analysis of PTM reuse and adoption trends.

3.2 Importance of Queryable PTM Metadata

PTM metadata has been applied for several tasks. In the realm of AI model management, the effective utilization of metadata plays a crucial role, such as helping with model auditing for assessing risks and ensuring responsible AI deployment [82]. Studies have shown that engineers often rely on various metadata types, such as evaluation metrics and hyperparameters, for informed model selection, underscoring their significance in the process. Existing techniques effectively extract key metadata, supported by research papers, including model names, datasets, and frameworks [95, 96]. However, these methods do not support extraction from model cards, and not work for a comprehensive list of metadata (*e.g.,* hyperparameters, model size, hardware specification) [21, 71]. The acquisition of extensive, queryable metadata types is crucial for enhancing model search, reuse, comparison, and composition [67].

The evolving landscape of model repositories presents new challenges for metadata extraction [54, 67]. The main problem is the greater number of kinds of artifacts in this context, and linking them together with corresponding GitHub repositories is academic papers are hard. Traditional methodologies have focused on model repositories on platforms like GitHub and academic papers [95, 96]. Some platforms have tried to link papers to the relevant code repositories and models together, such as PapersWithCode [7]. However, PTMs on model registries do not always link to GitHub projects and original research papers [54]. To address this gap in extracting metadata from model registries, we augment *PeaTMOSS* by leveraging state-of-the-art LLMs for metadata extraction. Capitalizing on the advanced capabilities of LLMs, we employ them to interpret and analyze model cards, effectively extracting pertinent metadata.

3.3 Open-Source PTM Datasets and Other Large-Scale Software Datasets

There are two existing PTM datasets: *PTMTorrent* [55] and *HFCommunity* [8]. Both provide data included in the Hugging Face model registry, offering insights into PTMs. However, both lack queryable PTM metadata and do not cover downstream applications. These limitations reduce the range of mining questions that can be posed. *PeaTMOSS* addresses both limitations by including additional content, *e.g.,* extracted metadata from model cards and links to downstream GitHub repositories.

There are also many large-scale open-source software datasets, such as GHTorrent [43], SOTorrent [12], and TravisTorrent [15]. These datasets offer long-term data availability and help researchers avoid API rate limits [43]. These datasets have been instrumental in improving our understanding of software engineering, *e.g.,* of practices in continuous integration [14], static analysis [108], software development [13, 24], and testing [32, 51].

To advance our understanding of software engineering practices in deep learning systems, a large-scale, open-source dataset similar to those in previous studies is essential [56]. Such a dataset should encompass extensive software and its associated, queryable metadata for research purposes [12, 43]. Additionally, it should include dependency information to effectively characterize the software supply chain and keep the data updated. *PeaTMOSS* has a broader scope by including downstream GitHub applications, updated metadata, and recent models. Notably, our dataset incorporates a substantial number of large language models (LLMs) like Llama 2, which were absent in prior datasets.

4 The PeatMOSS Dataset

This section summarizes and details the *PeaTMOSS* creation process.

4.1 Overview

We created the *PeaTMOSS* dataset to enable study about **P**re-**T**rained **M**odels in **O**pen- **S**ource **S**oftware. As illustrated by Figure 1, *PeaTMOSS* comprises snapshots of PTMs and open-source repositories utilizing PTMs, as well as a mapping of PTMs to projects. For both PTMs and GitHub projects, *PeaTMOSS* contains metadata (commits, issues, pull requests) and data (*e.g.,* model architecture and weights; `git` repositories), primarily collected in July-August 2023. Figure 3 presents a uniform schema for retrieving PTM and project metadata is provided to facilitate analysis of PTMs and their use in open-source software projects. Most information is indexed; some is stored as blobs.

PeaTMOSS contains the metadata of 281,638 PTM packages (281,276 from Hugging Face and 362 from PyTorch Hub), 28,575 GitHub projects that use PTMs as dependencies, and 44,337 links from these GitHub repositories to the PTMs they depend on.

The dataset can be accessed in two formats. The "metadata" version of *PeaTMOSS* is a 7.12 GB SQLite database. It contains the metadata of PTM packages and GitHub projects, and Globus links to their snapshots. The 48.2 TB "full" version has these snapshots: (1) the PTM package contents in each published version, and (2) `git` history of the main branches of the GitHub projects.

4.2 Dataset Creation Methodology

Here we outline the methodology employed to compile *PeaTMOSS*, detailing PTM collection in §4.2.1, and the approach for associating PTMs with downstream GitHub repositories in §4.2.2.

4.2.1 Collecting PTMs. First, we must identify the model registries whose PTMs we will collect. As discussed in §2.2, there are three types of model registries. Of these, only the open and gated types are open-source. For mining, we need registries that have APIs with recognizable signatures, allowing us to trace PTM-App dependencies (details in §4.2.2). Considering these criteria, we selected the most popular example from each open-source category that utilizes APIs. Thus, we included PTMs from Hugging Face (an open registry) and PyTorch Hub (a gated registry). Hugging Face contains far more PTMs than PyTorch Hub, which influenced several decisions we made in creating *PeaTMOSS*.

Our PTM data collection includes three parts: (1) We saved 14,296 PTM snapshots. This included the most popular PTM packages (*i.e.,* with over 50 downloads) on Hugging Face, and all PTMs on PyTorch Hub. This part of the data can provide a comprehensive view of PTM packages. (2) Among these "full" metadata, 44,337 links from the PTMs to the downstream GitHub repositories have been identified. This part of the data can be connected to downstream GitHub data

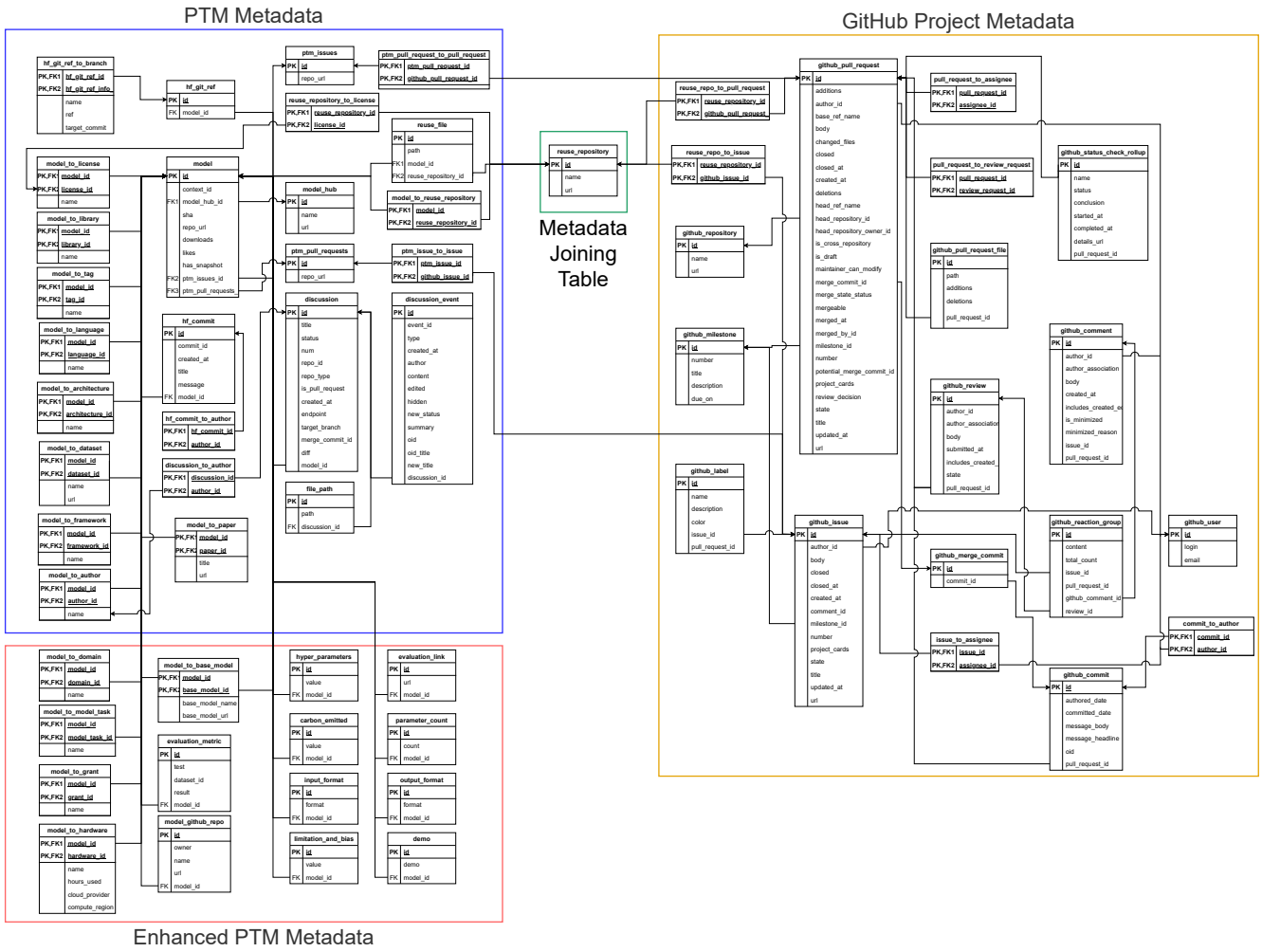

Figure 3: *PeaTMOSS* data schema. There are four regions: tables for PTMs (basic §4 and enhanced §5), tables for GitHub projects, and a table of PTM-Application dependency relations. Tables link to PTM and GitHub snapshots in a Globus share. Our artifact has a navigable version (§11).

and allows miners to analyze the relationship between them. (3) For all PTMs hosted on Hugging Face and PyTorch Hub, we retrieved their metadata, resulting in a total number of 281,638 PTM package metadata being included in *PeaTMOSS*.

Soundness and Completeness: *PeaTMOSS* is comprehensive in terms of popular PTM packages, as it includes snapshots of those with over 10,000 downloads on Hugging Face. This provides a full view of widely-used PTMs and their connections to downstream GitHub projects, facilitating in-depth analysis. Additionally, the dataset includes metadata from all other PTMs on Hugging Face, which can be used for metadata-based analyses. *PeaTMOSS* enhances the diversity of PTM data by incorporating PTM packages from PyTorch Hub, including all available model repositories and their associated pull requests and issues.

Implementation: Metadata is collected using an *Extract-Transform-Load (ETL)* pipeline for each model hub. We first *Extract* metadata from each model hub's API. Then we *Transform*, using this metadata to collect additional information (*e.g.*, following links to get packages backed by GitHub repositories). Data that fits the shared schema is placed in an intermediate representation, while other data is preserved as a blob. Results are *Loaded* into our database.

4.2.2 Collecting Downstream GitHub Repositories To enable research on *PTM-PTM* and *PTM-App* dependencies in open-source software projects, *PeaTMOSS* includes GitHub repositories that use at least one PTM from the two registries we captured. We obtained the 28,575 pertinent GitHub repositories that existed as of July 10, 2023. These repositories have an average of 201 stars.

For each of these 28,575 GitHub repositories, *PeaTMOSS* contains: (1) a full git clone; (2) all issues and associated metadata (as obtained through the GitHub CLI); and (3) all pull requests and associated metadata (via GitHub CLI). We link them to the PTMs they use that were collected in §4.2.1, to the extent possible with static analysis.

The main challenge for this part of the dataset is identifying the GitHub repositories that use PTMs. This task is non-trivial given the lack of standardized documentation or explicit labeling of PTM usage in repositories. We devised an approach to automatically

```python
from transformers import AutoTokenizer, AutoModelForMaskedLM

tokenizer = AutoTokenizer.from_pretrained("bert-base-multilingual-cased")
model = AutoModelForMaskedLM.from_pretrained("bert-base-multilingual-cased")
```

Figure 4: Example use of two HuggingFace PTMs. The code initializes a tokenizer (`AutoTokenizer`) and a model (`AutoModelForMaskedLM`) from the `transformers` library for a multilingual BERT model.

identify downstream GitHub repositories that depend on PTMs. There are four steps to our approach:

(Step 1) Signatures of PTM Use: The primary way to use PTMs from model hubs is through hub APIs. There are many model hub libraries that access these APIs to retrieve PTMs by name. Figure 4 gives an example of accessing PTMs from Hugging Face via its `Transformers` library.

We therefore define the *signature* of PTM usage in an application as the combination of (1) library import and (2) calls into that library to load a PTM. We specifically focus on signatures associated with Python libraries, as Python is the dominant language for PTM applications and almost all supported PTM loading libraries are written in Python [83, 92]. We manually identified libraries and signatures in the documentation for the two target model hubs, Hugging Face [34] and PyTorch [78]. In total, we found 474 signatures from 27 Python libraries that access these hubs.

(Step 2) Preliminary repository collection: We developed search patterns for each signature, and matched them against the content of files within GitHub repositories. We searched for signatures in public, non-fork, non-archived repositories. For this search, we used the `src` CLI tool from Sourcegraph, a popular code search engine that indexes GitHub repositories with ≥ 5 stars [91]. For example, a query for one of the signatures from the Diffusers library is: "`src select:file visibility:public count:all lang:Python content:'from diffusers' AND from_pretrained(`".

(Step 3) Static Analysis: As Sourcegraph's search feature relies on text-based patterns, it is possible that some of the search results are false positives (*e.g.,* signatures that occur in commented-out code). To mitigate this concern, we performed static analysis on the GitHub repositories from Step 2. This required some customization for each library. Given the number of signatures (474 signatures over 27 libraries), we focused on the most popular libraries. For PyTorch Hub, there are four libraries — `torchvision`, `torchaudio`, `torchtext`, and direct uses of `torch` — and we handle all associated signatures. For Hugging Face, there are 23 libraries. Figure 5 shows the distribution of usage: we used signatures for the top five libraries (`Transformers`, `SpaCy`, `Sentence-Transformers`, `Diffusers`, and `Timm`). These accounted for 96% of all downstream repositories that contain Hugging Face signatures according to our Sourcegraph search.

We performed static analysis using the Scalpel framework [64]. For each relevant source code file associated with a specific function signature, we construct an abstract syntax tree and extract the function calls contained within the file. Subsequently, we cross-reference the extracted functions with our predefined signatures which gives us a total of 28,575 repositories.

(Step 4) Mapping PTM-App relationship: Finally, we want to map which GitHub repositories depend on which PTMs. For

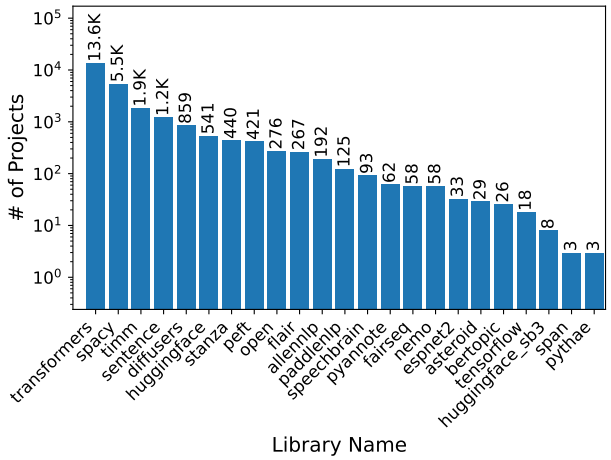

Figure 5: Number of projects that access PTMs from each Hugging Face library, as captured via Sourcegraph search. Note: Log scale.

the function calls from each signature that load PTMs (identified in step 1), we extracted the function arguments (one of which is the PTM name), enabling us to extract specific PTMs being used in downstream GitHub repositories. We identify repositories that statically call the collected PTMs — 15,129 GitHub repositories do so, loading 2,530 distinct PTMs. Note that a PTM may be used by multiple repositories, and a repository can use multiple PTMs.

5 Enhanced PeaTMOSS via Metadata Extraction

This section enhances *PeaTMOSS* by extracting indexed metadata from the unstructured metadata available in raw PTM packages. As discussed in §3.2, PTM metadata enables research and supports engineers' reuse process. Past work observed that PTM metadata is often available in model cards, but unstructured, hampering ecosystem analysis [54, 77, 95]. Our focus was on extracting metadata from Hugging Face PTM packages due to several reasons: (1) a larger quantity of PTM packages, (2) a larger quantity of mine-able documentation (model cards), and (3) the centralized accessibility of their model cards for collection purposes.

We propose to use Large Language Models (LLMs) to extract metadata from model cards. Recent studies have demonstrated the versatility of LLMs in various tasks, including information retrieval [19, 39]. LLMs are effective in the task of metadata extraction from scientific documents [30]. In this work, we use ChatGPT, a leading commercial LLM [1].

We identified desirable metadata through reviewing the literature and assessing available data in recent model cards, as shown in Table 2. Prior works on metadata extraction indicate metadata of interest. We supplemented those lists with metadata inspired by IBM's AI FactSheet [11], as well as observations from 50 recent model cards. These additional metadata include carbon emissions, model size, base model, limitation and biases, demonstration, grant/sponsorship information, and language.

Table 2: A list of PTM metadata mapped to the first paper that mentioned it. *PeaTMOSS* includes these fields plus more (last row of table). Our artifact shows the mapping of these fields to our schema.

Paper	Newly Introduced Metadata of Interest
Schelter *et al.*, 2017 [86]	Model name, model version, framework, tags, dataset name, dataset version, dataset statistic, data transform, input/output format, evaluation, training time, environment, hyperparameters, prediction metadata
Tsay *et al.*, 2020 [95]	Reference, domain, has README, uses Python, popularity
Li *et al.*, 2022 [67]	Model architecture, task, hardware
Tsay *et al.*, 2022 [96]	Description, code, training job, training output, provenance
PeaTMOSS	Carbon emitted, model size, license, base model, limitation and biases, demonstration, grant/sponsorship, language (NLP)

5.1 Prompt Design

Prompting provides the instructions to the LLM. We followed the prompt design flow proposed by Zamfirescu *et al.* [107], and outlined a structured approach for extracting and filling out a detailed metadata schema for models from Hugging Face. To enhance our pipeline's performance, we use iterative prompting to test random sampled models [50]. For metadata extracted with lower accuracy, we identified incorrect patterns, such as erroneous output formats and misleading results, to subsequently refine the corresponding prompts. Moreover, we meticulously recorded instances where the model erroneously inferred metadata, known as hallucinations [89], in the absence of relevant information. We also tracked cases where the model failed to extract information that was indeed present. Analyzing these outcomes enables us to pinpoint the metadata types that pose greater extraction challenges, thereby informing and refining our strategies in prompt engineering. The prompts for two pipelines are all available in §11.

The input prompt to our pipeline includes multiple components, as illustrated in Figure 6. The prefix prompt provides the domain and model background, setting extraction rules and schema adherence, with empty properties for absent document elements. The metadata prompt defines all the extraction requirements and formats for each metadata. The data schema is a formatted `json` file that used to store the extracted metadata. If domain and tasks are not pre-processed from model tags, we include domain and task prompts, specifying domains (*e.g.,* multimodal) and tasks (*e.g.,* text-to-image). For NLP models, language prompt is added to detail supported languages and extraction expectations (*e.g.,* Arabic, Chinese, Python).

5.2 LLM Pipeline Design

We designed two LLM pipelines, one optimizing monetary cost and the other accuracy. Figure 6 summarizes these pipelines.

Cheap Pipeline: By employing the Retrieval-Augmented Generation (RAG) strategy [63] to mitigate the token usage for each model card and using the more cost-effective `GPT-3.5-turbo`, we have developed an efficient "cheap" pipeline. The `GPT-3.5-turbo`'s token limit of 4,096 tokens per request necessitates a method to extract complete metadata in segmented operations. The RAG strategy helps the LLM incorporate relevant information from a knowledge base, providing contextual support and reducing the risk of generating inaccurate or speculative content. The RAG strategy reduces token usage, thus enhancing efficiency and reducing both computational and financial costs.

Accurate Pipeline: The accurate pipeline, utilizing `GPT-4-turbo`, has a substantial improvement on addressing the token limit issue, along with enhanced performance in data extraction [19]. An analysis of the token count across all model cards revealed that their lengths fell within the new token limit of `GPT-4-turbo` (128,000 tokens). Leveraging the advanced capabilities of GPT-4 [75], we streamlined our pipeline by removing the RAG component. This modification allowed for a more holistic understanding of each model card, thereby improving metadata extraction efficiency.

5.3 Evaluation

Sampling: Our initial evaluation required the selection of ground truth models, for which we analyzed the distribution of model tasks in the *PeaTMOSS* database. To achieve a representative sample, we employed stratified random sampling and sampled 50 models for evaluation. The models from different domains use different evaluation metrics so we want to cover most cases in our evaluation. In this approach, each model task functioned as a separate stratum. The sample size for each task was aligned with its proportional representation in the database. We focused on models that ranked among the top 100 in terms of downloads for each task, ensuring they were included in our database. We then carefully examined the information of these models by checking their model cards and manually created the ground truth metadata for them.

Accuracy: We selected accuracy as our primary metric for evaluating model performance, considering the context of manual assessment. This metric provides a straightforward and reliable method to evaluate the extraction. To calculate the overall accuracy, we tracked the frequency of successful metadata extractions matching our manual answer against the total number of extractions.

Results: Comparing our results with manually labeled data, the GPT-3.5-turbo based pipeline achieved an accuracy rate of 67.46%. This evaluation was conducted on a random sample of 50 model cards from the *PeaTMOSS* dataset. Notably, the average cost for the *cheap* pipeline was `$0.01/model`. In a subsequent re-evaluation using the identical dataset, the *accurate* pipeline exhibited a significant improvement in accuracy, reaching 94.39%. The average cost for the GPT-4-turbo pipeline was slightly higher, at `$0.03/model`. We have not evaluated the specific factors that enhanced GPT-4's performance in this context. However, its excellent performance led us to conclude the evaluation at this stage.

5.4 *PeaTMOSS* Enhancement

We enhanced the *PeaTMOSS* dataset by incorporating metadata obtained from the "accurate" LLM pipeline, focusing on models that have over 50 downloads — consistent with the model set for which we have collected snapshots. The enhancement not only add the metadata to our dataset, but also successfully identifies 8,829 *PTM-PTM* dependencies within the supply chain, pinpointing upstream base models linked to each model as specified in their model cards.

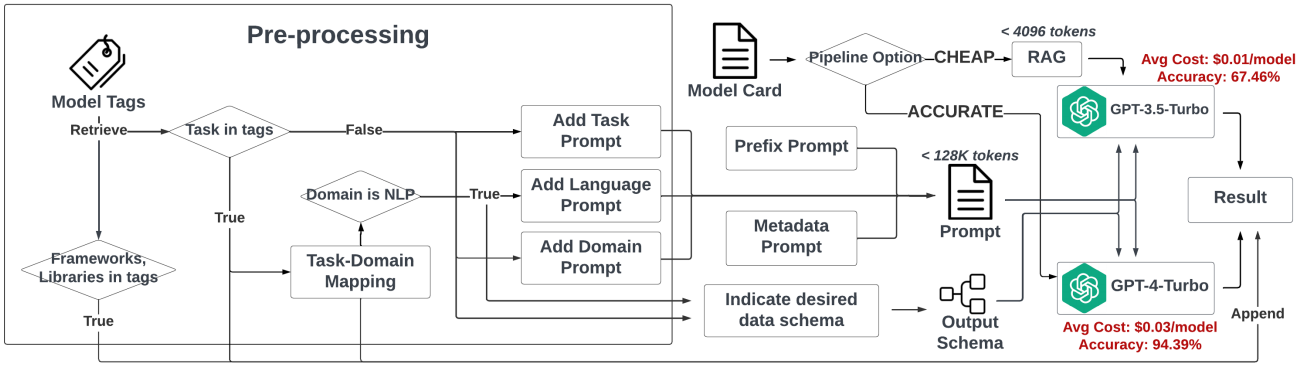

Figure 6: The "cheap" and "accurate" pipelines for metadata extraction. First the prompts are refined by the type of model. Then these prompts are applied to the model card. The cost-optimized "cheap" pipeline differs primarily by incorporating the Retrieval-Augmented Generation (RAG) framework [63] to reduce token count.

Running the "accurate" pipeline to extract these enhanced metadata took ~$400 and ~40 hours.

After metadata extraction, Figure 7 shows the percentage of available metadata types for PTM packages. Most models have metadata specifying libraries, domains, and model tasks, with 98.9% for libraries and slightly less for domains and tasks. Metadata on frameworks, licenses, datasets, base models, demonstration, and evaluation are also prevalent, although to a lesser extent. On the other hand, less than half of the models include metadata on provenance (*i.e.,* github_repo and papers). The data shows a significant absence of metadata concerning hyper-parameters, parameter count (*i.e.,* model size), hardware information, limitations, biases, and input/output formats, with these categories falling well below 40%. Less than 10% model cards indicate the grant/sponsorship information and carbon emission.

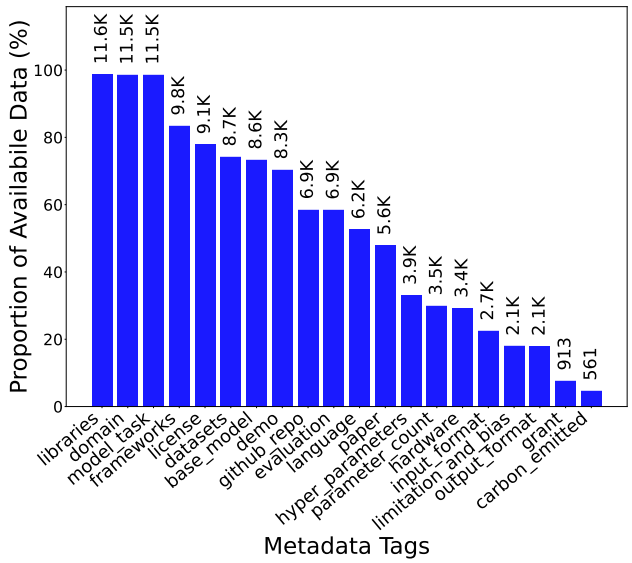

Figure 7: The proportion of available metadata for each category from Hugging Face model cards. Data is reported for the 11,975 models available at measurement time (Nov. 2023). About 17% of the sample (2,321 models) became unavailable between project initiation and this measurement.

6 *PeaTMOSS* Initial Data Analysis

We conduct some initial analysis of *PeaTMOSS* to illustrate its contents and measure the PTM supply chain. We report on the task domains of the PTMs (in aggregate and over time), PTM domains used by downstream GitHub repositories, and trends in model size.

Figure 8 presents the distribution of models across various problem domains in both Hugging Face and PyTorch Hub. It reveals that NLP models are predominant on Hugging Face (60.3%), whereas PyTorch Hub features a higher frequency of CV (56.1%) and Audio (27.6%) models.

Figure 9 displays the frequency of downstream GitHub repositories reusing PTM packages. It shows that NLP models are the most commonly reused on Hugging Face (75.4%), followed by Multimodal (17.4%) and CV (6.3%) models. Conversely, PyTorch Hub users predominantly utilize CV (96.0%), and only 2.23% of them use NLP models.

Figure 10 displays the creation frequency of Hugging Face PTM packages across various problem domains over time. The data indicates a predominance of Natural Language Processing (NLP) models, likely reflecting Hugging Face's initial focus on NLP PTMs. However, from August 2022 onward, packages for other domains have become more common. The jumps of NLP models in 2020 might relate to the rise of transformer family models during that time [74].

Figure 11 tracks the median model size (*i.e.,* parameter count) by different domain. There is a marked increase in the median size of NLP and multimodal PTMs, especially noticeable after March 2023. Meanwhile, the median parameter count for Audio and CV models has remained relatively stable.

7 Mining PTM-App License Compatibility

This section illustrates the use of *PeaTMOSS* through a simple mining study on software licensing. Software license information is important metadata for machine learning software [16, 60] and may promote responsible AI practices [23]. When using a PTM, an engineer should comply with its license. We ask: *(1) How do software licenses vary between PTMs and their downstream GitHub*

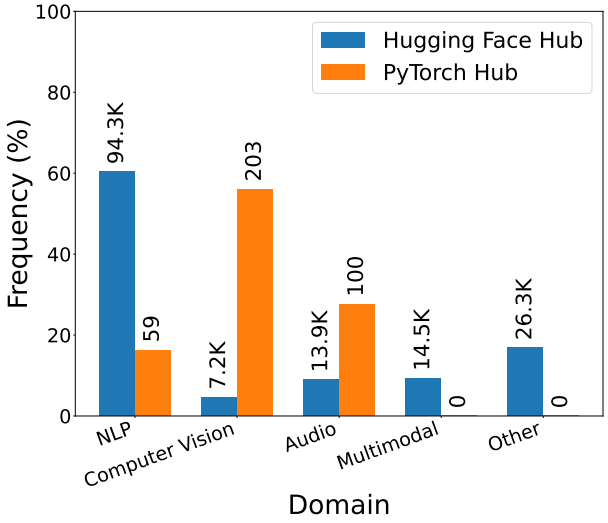

Figure 8: The distribution and frequency of domains across PTMs in the two hubs. For PyTorch Hub, we categorize labels such as research models, CUDA, and quantized models as "Other" for simplicity.

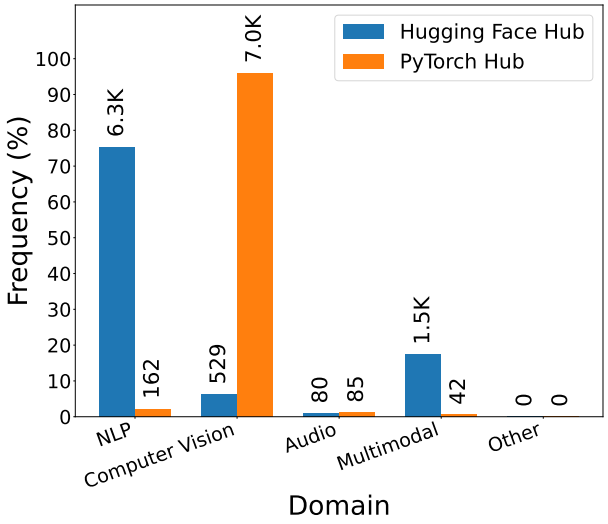

Figure 9: Distribution and frequency of downstream GitHub repositories by model task. Hugging Face contains far more PTMs than PyTorch Hub, but they see comparable use in GitHub repositories.

repositories? and *(2) How common are PTM-Application license incompatibilities?*

7.1 Background on Software Package Licensing

Licenses dictate the terms and conditions governing the reuse, modification, and redistribution of that software [61]. Licenses vary by the restrictions they place [40], *e.g.,* requiring derivative works to use a similar license (copyleft) or making the code freely available (public). Integrating software with different licenses is complex [5] and may result in legal issues [84, 90]. Studies of license incompatibility have been conducted in the Fedora Linux distribution [38], Android applications [97], and Java applications [37, 41]), as well as in multiple package ecosystems (*e.g.,* npm [80], RubyGems [69], and PyPI [104]). We ask similar questions in the PTM ecosystem.

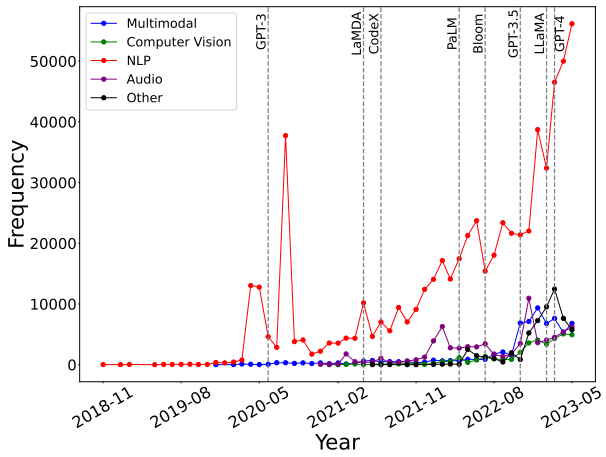

Figure 10: The frequency of Hugging Face PTMs for different problem domains, tracked over time. Vertical lines indicate events that may have caused the increase in parameters for NLP models.

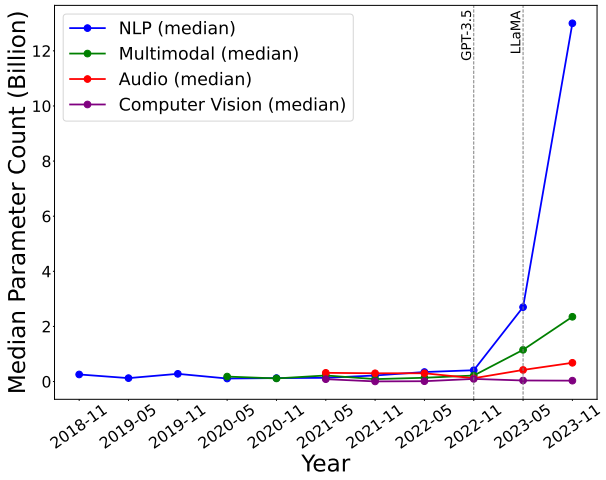

Figure 11: Number of parameters (median) over time. Vertical lines indicate landmarks of LLM models.

We treat licensing definitions in PTMs comparable to other software packages [38], with reuse (importing the PTM), modification (*e.g.,* fine-tuning a PTM), and redistribution (shipping the PTM in an application). Following prior work, we treat mismatches as cases when there are different levels of license restrictiveness [102].

7.2 License Measurement on *PeaTMOSS*

7.2.1 Method. In this analysis, we focus on the PTMs and GitHub projects in *PeaTMOSS* that are governed by a single license (7794, 54.5%). This model is simplistic [2] but aligns with GitHub's license API [40]. For PTMs, we use the license information from *PeaTMOSS* which was originally extracted using Hugging Face API from the model tags. For downstream GitHub repositories, *PeaTMOSS* also includes license information that we extracted using the `codescan` tool in imitation of GitHub's `licensee` tool (*e.g.,* referencing files such as `LICENSE.txt`). For PTM-Application dependencies, we use

the mapping given by *PeaTMOSS*. We manually measured license compatibility based on the Linux Foundation's OSS license compatibility table [94]. In license pairings where no legal compatibility analysis was available, *e.g.,* in the case of "no license" (43.42% of the downstream GitHub repositories), we omit an assessment.

7.2.2 Results. Figure 12 answers our questions in a Sankey diagram, on the part of *PeaTMOSS* for which we have PTM-Application dependencies — 2,530 PTMs used across 15,129 GitHub projects.

For license variation, we compare the left and right sides of Figure 12. The top-3 PTM licenses are Apache-2.0, MIT, and BSD-3-clause, while the top-3 GitHub repository licenses are MIT, Apache-2.0, and GPL-3.0-only. Many downstream GitHub repositories (43.42%) choose not to define a license ("no license" in the figure), instead operating under the default posture of Hugging Face (full reuse [48]) and GitHub (much stricter — copyright reserved to author [40]). In 25.61% of cases, the PTM-Application licenses are identical.

For license compatibility, Figure 12 indicates compatible licenses with blue flows, otherwise red. In 0.24% of PTM-Application dependencies, the licenses are incompatible. In *PeaTMOSS*, this is the result of copyleft provisions in the PTM's license that are not honored by the Application.

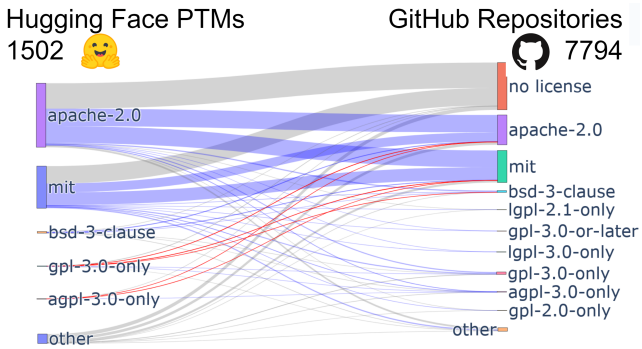

Figure 12: Sankey Diagram for license compatibility. Flows represent the licenses of PTMs and the downstream GitHub repositories that use them. Blue flows are compatible, red are not. Grey flows (47.33% of pairs) represent license pairs that have not been analyzed by the Linux Foundation — this is primarily caused by GitHub repositories lacking an explicit license.

8 Threats to Validity

Users of *PeaTMOSS* will inherit three types of threats to validity.

Construct Validity. In the *PeaTMOSS* dataset, a key construct threat is the exclusion of conceptual reuse of PTMs [26], which may limit our understanding of PTM usage. In our license analysis (§7), we assume that each project has one license, which aligns with GitHub's model (License API) but is imperfect. Additionally, our metadata extraction method is supported by LLMs, which can pose threats on the reliability of our dataset. To mitigate this, we conducted an evaluation using stratified random sampling, observing accuracy of 94% (§5.3).

Internal Validity. Internal validity threats in our study stem from the possibility of selection bias in curating PTMs and GitHub projects, as well as the changeable nature of repository contents over time. Specifically, our LLM pipeline evaluation might be biased since it is based on a sample of only 50 models from Hugging Face. To mitigate this, we employed a stratified sampling technique. Additionally, when identifying the mapping from PTM to downstream GitHub repositories, our reliance on keyword searches on GitHub could miss other reuse signatures, potentially limiting the comprehensiveness of our findings. To mitigate this, we employ distinct methodologies to manually gather the relevant signatures from Hugging Face and PyTorch Hub.

Another internal threat exists in our work due to the potential inaccuracies in the metadata extraction process from unstructured model cards. To mitigate this threat, we evaluated our extraction pipeline against a set of 50 manually-labeled model cards and found high accuracy. Moreover, we implemented a confidence threshold mechanism within our LLM extraction process. If the confidence level is below the set threshold, the data is earmarked for manual review, thereby improving the reliability of our dataset.

External Validity. External validity threats are present due to the dataset primarily sourcing from Hugging Face and PyTorch Hub, which might not represent all PTM usage scenarios. To mitigate this threat, our dataset was designed to be expandable. We built our database in a flexible, modular way and provided clear, detailed documentation. This makes it straightforward to add new information to the database as needed. Another threat is present due to the selection of five libraries for the Hugging Face PTM downstream application collection. To mitigate the threat we check that 96% of the primarily collected applications for Hugging Face PTMs belong to those five libraries. This fulfills the representativeness of our dataset. Additionally, we acknowledge that there is another external threat on the dynamic nature of PTM data. The dataset will need to be updated — we evaluated our data collection programs on multiple sites with comprehensive documentation provided for ongoing and future research.

9 Future Work

Every dataset can be improved. We highlight two enhancements for *PeaTMOSS*. First, the current *PTM-App* mapping relies solely on static analysis; a valuable extension would be to identify dynamic PTM usage. Second, a deeper dive into the PTM supply chain would categorize the patterns of reuse in GitHub downstream repositories, such as direct loading versus fine-tuning or extending a model. Adding these patterns would enrich the dataset.

PeaTMOSS enables many lines of research. We highlight three lines in Table 3. The first line of research studies the Pre-Trained Model portion (*PTM*). The second line of research studies the GitHub portion of the dataset (*GH*). The third line integrates both parts (*I*).

10 Conclusion

Pre-Trained Models (PTMs) offer state-of-the-art performance in various domains, and are being incorporated into many computing systems. PTMs represent a new frontier for mining software repositories, but the community lacks a comprehensive dataset. To enable PTM mining, this paper presents the *PeaTMOSS* dataset, a collection of PTM metadata, PTM snapshots, downstream GitHub repositories that use PTMs, and mappings between PTMs and the repositories that use them. To augment the data available from PTM registries and GitHub APIs, we developed an automated process

Table 3: Example lines of research for researchers to investigate, phrased as research questions. These questions are divided into three groups. The first group uses the Pre-Trained Model portion of the dataset (*PTM*). The second group of questions makes use of the GitHub portion of the dataset (*GH*). The third group asks questions that require Integrating both parts of the dataset (*I*).

Research question	Related work
PTM-1: What factors predict the popularity of a PTM? Intuition suggests that performance aspects such as accuracy and latency may dominate; what is the role played by factors such as software engineering quality?	[17, 54, 68]
PTM-2: What naming conventions do PTMs follow? Are they consistent enough (within an architecture family? across families?) to support engineers looking for similar models?	[44, 53]
PTM-3: PTM authors may reuse each others' work, *e.g.,* building off of model checkpoints or incorporating architectural building blocks. Is this forking, or a new form of software exchange? What is the phylogeny of the families of PTMs?	[56]
PTM-4: There are many concerns about DNNs with unexpected or malicious behavior. How common are such DNNs?	[45, 54, 100, 101]
PTM-5: How to improve safety and security of model infrastructure, such as serialization formats and interoperability?	[25, 49]
GH-1: What kinds of defects are opened related to PTM use in the GitHub projects? How do these defects differ from defects opened on other aspects of the GitHub projects?	[72]
GH-2: What do developers on GitHub discuss related to PTM use, *e.g.,* in the body text of issues and pull requests? What are developers' sentiments regarding PTM use? Do the people issuing pull requests for PTMs have the right expertise?	[85, 106]
GH-3: How often do developers change the PTM used to implement a feature? What factors influence this?	[28]
GH-4: PTMs can underpin, enhance, or replace features implemented with traditional code. How common are these three modes of PTM adoption? How do PTMs subsequently affect the feature's failure modes?	[66, 73, 79, 111]
I-1: It can be difficult to interpret model popularity numbers by download rates. To what extent does a PTM's download rates correlate with the number of GitHub projects that rely on it, or the popularity of the GitHub projects?	[35]
I-2: What are code smells for PTMs in the downstream GitHub repositories, and how do they affect these projects?	[20, 98, 110]
I-3: What are application engineers' testing practices for their PTM-enabled features? Do these vary based on the project's purpose, or the task delegated to the PTM? How do testing practices acknowledge and address PTM stochasticity (*e.g.,* "flakiness")?	[18, 31, 65, 73]
I-4: How often do PTM application engineers update their PTM dependencies, *e.g.,* due to (1) PTM deprecation, (2) PTM improvement, or (3) PTM antiquation (newer, better model)? What is the typical technical lag for such updates?	[47, 99, 109]
I-5: What are the characteristics of issue reports on PTM packages, *e.g.,* in terms of the kinds of questions asked, responsiveness of maintainers, issue density, and issue staleness? How do these attributes differ from issue reports in GitHub repositories?	[52, 105]
I-6: What are the software signing requirements for PTMs? What are effective signatures for PTMs and training regimes?	[87]

to extract and standardize PTM metadata, enhancing the dataset's utility. To demonstrate applications of *PeaTMOSS*, we present the first detailed statistics of the PTM supply chain, and examine software license inconsistencies between PTMs and their dependent projects. For future work, we propose thirteen distinct research questions along three lines of research: studies of PTMs, studies of downstream use on GitHub, and studies that integrate data on PTMs and their dependents.

11 Data Availability

The source code associated with this research is available at https://github.com/PurdueDualityLab/PeaTMOSS-Artifact, along with a demo version of the dataset. The full *PeaTMOSS* dataset is stored on our organization's archival-grade storage system and accessible through Globus at https://transfer.rcac.purdue.edu/file-manager?origin_id=ff978999-16c2-4b50-ac7a-947ffdc3eb1d&jorigin_path=%2F.

Acknowledgments

This work was supported by gifts from Google and Cisco; NSF awards #2107230, #2229703, #2107020, and #2104319; and by the Faculty Research Participation Program at Argonne National Laboratory. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2019-05071. This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357. We thank Purdue's Rosen Center for Advanced Computing (RCAC) for ongoing support in hosting the dataset. We thank A. Raghav, A. Qi, and Y. Mehta for their assistance, and members of the Purdue Duality Lab for their feedback on the manuscript.

References

[1] [n. d.]. ChatGPT. https://chat.openai.com
[2] [n. d.]. CodeScan: Code Quality and Security for Salesforce. https://www.codescan.io.
[3] [n. d.]. npm: Build Amazing Things. https://www.npmjs.com/.
[4] [n. d.]. PyPI: The Python Package Index. https://pypi.org/.
[5] 2022. License compatibility. https://en.wikipedia.org/wiki/License_compatibility
[6] 2023. NGC Catalog - GPU-optimized AI, Machine Learning, & HPC Software. https://catalog.ngc.nvidia.com.
[7] Meta AI. 2024. Papers With Code. https://paperswithcode.com/about
[8] Adem Ait, Javier Luis Cánovas Izquierdo, and Jordi Cabot. 2023. HFCommunity: A Tool to Analyze the Hugging Face Hub Community. In *SANER'23*. IEEE.
[9] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, and Harald Gall. 2019. Software Engineering for Machine Learning: A Case Study. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*.
[10] Antreas Antoniou, Amos Storkey, and Harrison Edwards. 2017. Data augmentation generative adversarial networks. *arXiv:1711.04340* (2017).
[11] Matthew Arnold, Rachel KE Bellamy, Michael Hind, Stephanie Houde, Sameep Mehta, Aleksandra Mojsilović, Ravi Nair, K Natesan Ramamurthy, Alexandra Olteanu, David Piorkowski, et al. 2019. FactSheets: Increasing trust in AI services through supplier's declarations of conformity. *IBM Journal of Research and Development* 63, 4/5 (2019), 6–1.
[12] Sebastian Baltes. 2018. SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts. In *Internat'l. Conf. on Mining Software Repos. (MSR)*.
[13] Sebastian Baltes and Stephan Diehl. 2019. Usage and attribution of Stack Overflow code snippets in GitHub projects. *EMSE* (2019).
[14] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub. In *International Conference on Mining Software Repositories (MSR)*.
[15] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration. In *International Conference on Mining Software Repositories (MSR)*.
[16] Aaditya Bhatia, Ellis E. Eghan, Manel Grichi, William G. Cavanagh, Zhen Ming Jiang, and Bram Adams. 2023. Towards a change taxonomy for machine learning pipelines: Empirical study of ML pipelines and forks related to academic publications. *EMSE* 28 (2023). https://doi.org/10.1007/s10664-022-10282-8
[17] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the factors that impact the popularity of GitHub repositories. In *Internat'l. Conf. on Software Maintenance and Evolution (ICSME)*. IEEE.
[18] Houssem Ben Braiek and Foutse Khomh. 2020. On testing machine learning programs. *Journal of Systems and Software (JSS)* 164 (2020), 110542.
[19] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv:2303.12712
[20] Nicolás Cardozo, Ivana Dusparic, and Christian Cabrera. 2023. Prevalence of Code Smells in Reinforcement Learning Projects. *arXiv:2303.10236* (2023).
[21] Joel Castaño, Silverio Martínez-Fernández, Xavier Franch, and Justus Bogner. 2023. Analyzing the Evolution and Maintenance of ML Models on Hugging Face. *arXiv* (2023). https://arxiv.org/abs/2311.13380
[22] Joel Castaño, Silverio Martínez-Fernández, Xavier Franch, and Justus Bogner. 2023. Exploring the Carbon Footprint of Hugging Face's ML Models: A Repository Mining Study. *arXiv* (2023). https://arxiv.org/pdf/2305.11164.pdf
[23] Danish Contractor, Daniel McDuff, Julia Katherine Haines, Jenny Lee, Christopher Hines, Brent Hecht, Nicholas Vincent, and Hanlin Li. 2022. Behavioral use licensing for responsible AI. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 778–788.
[24] Valerio Cosentino, Javier L Cánovas Izquierdo, and Jordi Cabot. 2017. A systematic mapping study of software development with GitHub. *IEEE Access* (2017).
[25] Yaniv David, Neophytos Christou, Andreas D. Kellas, Vasileios P. Kemerlis, and Junfeng Yang. 2024. QUACK: Hindering Deserialization Attacks via Static Duck Typing. In *the Network and Distributed System Security Symposium (NDSS)*.
[26] James C. Davis, Purvish Jajal, Wenxin Jiang, Taylor R. Schorlemmer, Nicholas Synovic, and George K. Thiruvathukal. 2023. Reusing Deep Learning Models: Challenges and Directions in Software Engineering. In *Proceedings of the IEEE John Vincent Atanasoff Symposium on Modern Computing (JVA'23)*.
[27] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* 108, 4 (April 2020), 485–532.
[28] Malinda Dilhara, Ameya Ketkar, and Danny Dig. 2021. Understanding Software-2.0: A Study of Machine Learning library usage and evolution. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 4 (2021), 1–42.
[29] Parijat Dube, Bishwaranjan Bhattacharjee, Siyu Huo, Patrick Watson, and Brian Belgodere. 2019. Automatic Labeling of Data for Transfer Learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.
[30] Alexander Dunn, John Dagdelen, Nicholas Walker, Sanghoon Lee, Andrew S Rosen, Gerbrand Ceder, Kristin Persson, and Anubhav Jain. 2022. Structured information extraction from complex scientific text with fine-tuned large language models. *arXiv preprint arXiv:2212.05238* (2022).
[31] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli. 2019. Understanding flaky tests: The developer's perspective. In *ACM Join Meeting on European Software Eng. Conf. and Sympos. on the Found. of Software Eng.(ESEC/FSE)*.
[32] D. Elsner, F. Hauer, A. Pretschner, and S. Reimer. 2021. Empirically evaluating readily available information for regression test optimization in continuous integration. In *International Symposium on Software Testing and Analysis (ISSTA)*.
[33] Hugging Face. 2021. Hugging Face – The AI community building the future. https://huggingface.co/
[34] Hugging Face. 2023. Hugging Face Hub Library Documentation. https://github.com/HuggingFace/hub-docs/blob/main/js/src/lib//interfaces/Libraries.ts
[35] Yuanrui Fan, Xin Xia, David Lo, Ahmed E Hassan, and Shanping Li. 2021. What makes a popular academic AI repository? *EMSE* 26 (2021), 1–35.
[36] Jonas Geiping and Tom Goldstein. 2023. Cramming: Training a Language Model on a single GPU in one day. In *International Conf. on Machine Learning (ICML)*.
[37] Daniel German and Massimiliano Di Penta. 2012. A method for open source license compliance of java applications. *IEEE software* 29, 3 (2012), 58–63.
[38] Daniel M German, Massimiliano Di Penta, and Julius Davies. 2010. Understanding and auditing the licensing of open source software distributions. In *2010 IEEE 18th International Conference on Program Comprehension*. IEEE, 84–93.
[39] Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. ChatGPT Outperforms Crowd-Workers for Text-Annotation Tasks. *Proceedings of the National Academy of Sciences* 30 (July 2023), e2305016120. http://arxiv.org/abs/2303.15056
[40] GitHub. 2022. GitHub REST API Documentation on Licenses. https://docs.github.com/en/rest/licenses/licenses. API version: 2022-11-28.
[41] Yaroslav Golubev, Maria Eliseeva, Nikita Povarov, and Timofey Bryksin. 2020. A study of potential code borrowing and license violations in java projects on github. In *International Conference on Mining Software Repositories (MSR)*.
[42] L. Gong, J. Zhang, M. Wei, H. Zhang, and Z. Huang. 2023. What is the intended usage context of this model? An exploratory study of pre-trained models on various model repositories. *TOSEM* 32, 3 (2023), 1–57.
[43] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: Github's data from a firehose. In *Internat'l Working Conf. on Mining Software Repositories (MSR)*.
[44] Remo Gresta, Vinicius Durelli, and Elder Cirilo. 2023. Naming Practices in Object-oriented Programming: An Empirical Study. *Journal of Software Engineering Research and Development* (2023), 5–1.
[45] Shangwei Guo, Chunlong Xie, Jiwei Li, Lingjuan Lyu, and Tianwei Zhang. 2022. Threats to pre-trained language models: Survey and taxonomy. *arXiv preprint arXiv:2202.06862* (2022).
[46] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, et al. 2021. Pre-trained models: Past, present and future. *AI Open* 2 (2021), 225–250.
[47] André Hora, Romain Robbes, Marco Tulio Valente, Nicolas Anquetil, Anne Etien, and Stéphane Ducasse. 2018. How do developers react to API evolution? A large-scale empirical study. *Software Quality Journal* 26 (2018), 161–191.
[48] Hugging Face. 2023. Repositories and Licenses. https://huggingface.co/docs/hub/repositories-licenses.
[49] Purvish Jajal, Wenxin Jiang, Arav Tewari, Joseph Woo, Yung-Hsiang Lu, George K Thiruvathukal, and James C Davis. 2023. Analysis of Failures and Risks in Deep Learning Model Converters: A Case Study in the ONNX Ecosystem. *arXiv* (2023). https://arxiv.org/abs/2303.17708
[50] Susmit Jha, Sumit Kumar Jha, Patrick Lincoln, Nathaniel D Bastian, Alvaro Velasquez, and Sandeep Neema. 2023. Dehallucinating large language models using formal methods guided iterative prompting. In *2023 IEEE International Conference on Assured Autonomy (ICAA)*. IEEE, 149–152.
[51] Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. Cure: Code-aware neural machine translation for automatic program repair. In *International Conference on Software Engineering (ICSE'21)*. IEEE, 1161–1173.
[52] W. Jiang, V. Banna, N. Vivek, A. Goel, N. Synovic, G.K. Thiruvathukal, and J.C. Davis. 2023. Challenges and practices of deep learning model reengineering: A case study on computer vision. *arXiv* (2023).
[53] Wenxin Jiang, Chingwo Cheung, George K. Thiruvathukal, and James C. Davis. 2023. Exploring Naming Conventions (and Defects) of Pre-trained Deep Learning Models in Hugging Face and Other Model Hubs. *arXiv:2310.01642* (2023).
[54] Wenxin. Jiang, Nicholas. Synovic, Matt. Hyatt, Taylor R. Schorlemmer, Rohan. Sethi, Yung-Hsiang Lu, George K. Thiruvathukal, and James C. Davis. 2023. An Empirical Study of Pre-Trained Model Reuse in the Hugging Face Deep Learning Model Registry. In *ICSE*.
[55] W. Jiang, N. Synovic, P. Jajal, T.R. Schorlemmer, A. Tewari, B. Pareek, G.K. Thiruvathukal, and J.C. Davis. 2023. PTMTorrent: A Dataset for Mining Open-source Pre-trained Model Packages. *MSR* (2023).
[56] W. Jiang, N. Synovic, R. Sethi, A. Indarapu, M. Hyatt, T.R. Schorlemmer, G.K. Thiruvathukal, and J.C. Davis. 2022. An Empirical Study of Artifacts and Security Risks in the Pre-Trained Model Supply Chain. In *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED)*.

[57] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. 2021. Ammus: A survey of transformer-based pretrained models in natural language processing. *arXiv* (2021).
[58] Andrej Karpathy. 2017. Software 2.0. https://karpathy.medium.com/software-2-0-a64152b37c35. (2017), 1–8.
[59] A. Kathikar, A. Nair, B. Lazarine, A. Sachdeva, and S. Samtani. 2023. Assessing the vulnerabilities of the open-source artificial intelligence (AI) landscape: A large-scale analysis of the Hugging Face platform. In *Intern. Conf. on Intelligence and Security Informatics*.
[60] P. Kuckertz, J. Göpfert, O. Karras, D. Neuroth, J. Schönau, R. Pueblas, S. Ferenz, F. Engel, N. Pflugradt, J.M. Weinand, A. Nieße, S. Auer, and D. Stolten. 2023. A Metadata-Based Ecosystem to Improve the FAIRness of Research Software. http://arxiv.org/abs/2306.10620
[61] Andrew M St Laurent. 2004. *Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software.* O'Reilly Media.
[62] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444. https://doi.org/10.1038/nature14539
[63] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Adv. in Neural Information Processing Systems (NeurIPS)* (2020).
[64] Li Li, Jiawei Wang, and Haowei Quan. 2022. Scalpel: The Python Static Analysis Framework. *arXiv preprint arXiv:2202.11840* (2022).
[65] S. Li, J. Guo, J.G. Lou, M. Fan, T. Liu, and D. Zhang. 2022. Testing Machine Learning Systems in Industry: An Empirical Study. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 263–272.
[66] Y. Li, Z. Zhang, B. Liu, Z. Yang, and Y. Liu. 2021. ModelDiff: Testing-based DNN similarity comparison for model reuse detection. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. 139–151.
[67] Ziyu Li, Rihan Hai, Alessandro Bozzon, and Asterios Katsifodimos. 2022. Metadata Representations for Queryable ML Model Zoos. arXiv:2207.09315
[68] Caroline Lima and Andre Hora. 2020. What are the characteristics of popular APIs? A large-scale study on Java, Android, and 165 libraries. *Software Quality Journal* 28, 2 (2020), 425–458.
[69] Ilyas Saïd Makari, Ahmed Zerouali, and Coen De Roover. 2022. Prevalence and Evolution of License Violations in npm and RubyGems Dependency Networks. In *International Conference on Software and Software Reuse*. Springer, 85–100.
[70] Pedro Marcelino. 2022. Transfer learning from pre-trained models. https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751
[71] Diego Montes, Pongpatapee Peerapatanapokin, Jeff Schultz, Chengjun Guo, Wenxin Jiang, and James C Davis. 2022. Discrepancies among pre-trained deep neural networks: a new threat to model zoo reliability. In *ESEC/FSE-IVR track*.
[72] Mohammad Mehdi Morovati, Amin Nikanjam, Florian Tambon, Foutse Khomh, and Zhen Ming. 2023. Bug Characterization in Machine Learning-based Systems. *arXiv* (2023). https://arxiv.org/abs/2307.14512
[73] Nadia Nahar, Haoran Zhang, Grace Lewis, Shurui Zhou, and Christian Kästner. 2023. A Dataset and Analysis of Open-Source Machine Learning Products. *arXiv preprint arXiv:2308.04328* (2023).
[74] NLPlanet. 2021. A Brief Timeline of NLP: From Bag of Words to the Transformer Family. *Medium* (2021). https://medium.com/nlplanet/a-brief-timeline-of-nlp-from-bag-of-words-to-the-transformer-family-7caad8bbba56
[75] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
[76] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon Emissions and Large Neural Network Training. https://doi.org/10.48550/arXiv.2104.10350
[77] David Piorkowski, Michael Hind, and John Richards. 2023. Quantitative ai risk assessments: Opportunities and challenges. *arXiv* (2023).
[78] Pytorch. 2021. PyTorch Hub. https://pytorch.org/hub/
[79] Binhang Qi, Hailong Sun, Xiang Gao, Hongyu Zhang, Zhaotian Li, and Xudong Liu. 2023. Reusing Deep Neural Network Models through Model Re-engineering. In *International Conference on Software Engineering (ICSE)*.
[80] Shi Qiu, Daniel M German, and Katsuro Inoue. 2021. Empirical study on dependency-related license violation in the javascript package ecosystem. *Journal of Information Processing* 29 (2021), 296–304.
[81] Saidur Rahman, Emilio River, Foutse Khomh, Yann Gal Guhneuc, and Bernd Lehnert. 2019. Machine learning software engineering in practice: An industrial case study. *arXiv preprint* (2019). https://doi.org/10.48550/arXiv.1906.07154
[82] Inioluwa Deborah Raji, Andrew Smart, Rebecca N White, et al. 2020. Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In *The 2020 conference on fairness, accountability, and transparency*.
[83] Sebastian Raschka, Joshua Patterson, and Corey Nolet. 2020. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information* 11, 4 (2020), 193.
[84] Lawrence Rosen. 2005. Open source licensing. *Software Freedom and Intellectual Property Law* (2005).
[85] A. Sajadi, K. Damevski, and P. Chatterjee. 2023. Interpersonal Trust in OSS: Exploring Dimensions of Trust in GitHub Pull Requests. In *International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*.
[86] S. Schelter, J.H. Boese, J. Kirschnick, T. Klein, and S. Seufert. 2017. Automatically tracking metadata and provenance of machine learning experiments. In *Conference on Neural Information Processing Systems (NeurIPS)*.
[87] Taylor R Schorlemmer, Kelechi G Kalu, Luke Chigges, Kyung Myung Ko, et al. 2024. Signing in Four Public Software Package Registries: Quantity, Quality, and Influencing Factors. arXiv:2401.14635
[88] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang. 2023. HuggingGPT: Solving AI tasks with ChatGPT and its Friends in HuggingFace. *arXiv:2303.17580* (2023).
[89] S.S. Sohail, F. Farhat, Y. Himeur, M. Nadeem, D.O. Madsen, Y. Singh, S. Atalla, and W. Mansoor. 2023. Decoding ChatGPT: a taxonomy of existing research, current challenges, and possible future directions. *Journal of King Saud University-Computer and Information Sciences* (2023).
[90] M. Sojer, O. Alexy, S. Kleinknecht, and J. Henkel. 2014. Understanding the drivers of unethical programming behavior: The inappropriate reuse of internet-accessible code. *Journal of Management Info. Systems* 31, 3 (2014), 287–325.
[91] Sourcegraph. 2023. https://docs.sourcegraph.com/cli
[92] Xin Tan, Kai Gao, Minghui Zhou, and Li Zhang. 2022. An exploratory study of deep learning supply chain. In *Intern. Conf. on Software Engineering (ICSE)*.
[93] Mina Taraghi, Gianolli Dorcelus, Armstrong Foundjem, Florian Tambon, and Foutse Khomh. 2024. Deep Learning Model Reuse in the HuggingFace Community: Challenges, Benefit and Trends. arXiv:2401.13177
[94] The Linux Foundation. 2019. Fulfilling Open Source License Obligations: Can Checklists Help? https://events19.linuxfoundation.org/wp-content/uploads/2018/07/OSLS-2019-Fulfilling-Open-Source-license-obligations-Can-checklists-help.pdf.
[95] J. Tsay, A. Braz, M. Hirzel, A. Shinnar, and T. Mummert. 2020. AIMMX: Artificial Intelligence Model Metadata Extractor. In *Mining Softw. Repos. (MSR)*.
[96] J. Tsay, M. Braz, A.and Hirzel, A. Shinnar, and T. Mummert. 2022. Extracting enhanced artificial intelligence model metadata from software repositories. *Empirical Software Engineering* 27, 7 (Dec. 2022), 176.
[97] S. Van Der Burg, E. Dolstra, S. McIntosh, J. Davies, D.M. German, and A. Hemel. 2014. Tracing software build processes to uncover license compliance inconsistencies. In *Automated software engineering (ASE)*. 731–742.
[98] Bart Van Oort, Luís Cruz, Maurício Aniche, and Arie Van Deursen. 2021. The prevalence of code smells in machine learning projects. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 1–8.
[99] Chengcheng Wan, Shicheng Liu, Henry Hoffmann, Michael Maire, and Shan Lu. 2021. Are machine learning cloud apis used correctly?. In *ICSE*.
[100] Shuo Wang, Surya Nepal, Carsten Rudolph, Marthie Grobler, Shangyu Chen, and Tianle Chen. 2022. Backdoor Attacks Against Transfer Learning With Pre-Trained Deep Learning Models. *IEEE Transactions on Services Computing* 15, 3 (May 2022), 1526–1539. https://doi.org/10.1109/TSC.2020.3000900
[101] Zhi Wang, Chaoge Liu, Xiang Cui, Jie Yin, and Xutong Wang. 2022. EvilModel 2.0: Bringing Neural Network Models into Malware Attacks. *Computers & Security* (2022). https://doi.org/10.1016/j.cose.2022.102807
[102] Thomas Wolter, Ann Barcomb, Dirk Riehle, and Nikolay Harutyunyan. 2023. Open source license inconsistencies on github. *ACM TOSEM* 32, 5 (2023).
[103] Jun Xia, Yanqiao Zhu, Yuanqi Du, Y Liu, and SZ Li. 2023. A Systematic Survey of Chemical Pre-trained Models. International Joint Conference on Artificial Intelligence (IJCAI'23).
[104] Weiwei Xu, Hao He, Kai Gao, and Minghui Zhou. 2023. Understanding and Remediating Open-Source License Incompatibilities in the PyPI Ecosystem. *arXiv preprint arXiv:2308.05942* (2023).
[105] Zhou Yang, Chenyu Wang, Jieke Shi, Thong Hoang, Pavneet Kochhar, Qinghua Lu, Zhenchang Xing, and David Lo. 2023. What Do Users Ask in Open-Source AI Repositories? An Empirical Study of GitHub Issues. *arXiv* (2023).
[106] Likang Yin and Vladimir Filkov. 2020. Team discussions and dynamics during DevOps tool adoptions in OSS projects. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 697–708.
[107] J.D. Zamfirescu-Pereira, R.Y. Wong, B. Hartmann, and Q. Yang. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Conference on Human Factors in Computing Systems (CHI)*.
[108] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta. 2017. How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines. In *International Conference on Mining Software Repositories (MSR)*.
[109] Ahmed Zerouali, Eleni Constantinou, Tom Mens, Gregorio Robles, and Jesús González-Barahona. 2018. An empirical analysis of technical lag in npm package dependencies. In *International Conference on Software Reuse*. Springer, 95–110.
[110] Haiyin Zhang, Luís Cruz, and Arie Van Deursen. 2022. Code smells for machine learning applications. In *Internat'l Conf. on AI Eng.: Software Eng. for AI*. 217–228.
[111] Ziqi Zhang, Yuanchun Li, Jindong Wang, Bingyan Liu, Ding Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2022. ReMoS: reducing defect inheritance in transfer learning via relevant model slicing. In *ICSE'22*.
[112] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2020), 43–76.