10-5-2023

PeaTMOSS: Mining Pre-Trained Models in Open-Source Software

Wenxin Jiang
Purdue University, jiang784@purdue.edu

Jason Jones
Purdue University

Jerin Yasmin
Queen's University - Kingston, Ontario

Nicholas Synovic
Loyola University Chicago, nsynovic@luc.edu

Rajiv Sashti
Purdue University

See next page for additional authors

PeaTMOSS: Mining Pre-Trained Models in Open-Source Software

Wenxin Jiang[1*], Jason Jones[1*], Jerin Yasmin[2*], Nicholas Synovic[3], Rajeev Sashti[1], Sophie Chen[4], George K. Thiruvathukal[3], Yuan Tian[2], James C. Davis[1]
Purdue University[1] ; Queen's University[2] ; Loyola University–Chicago[3] ; and University of Michigan–Ann Arbor[4]
West Lafayette, Indiana, USA[1] ; Kingston, Ontario, CA[2] ; Chicago, Illinois, USA[3] ; and Ann Arbor, Michigan, USA[4]

***Abstract*—Developing and training deep learning models is expensive, so software engineers have begun to reuse pre-trained deep learning models (PTMs) and fine-tune them for downstream tasks. Despite the wide-spread use of PTMs, we know little about the corresponding software engineering behaviors and challenges.**

To enable the study of software engineering with PTMs, we present the `PeaTMOSS` dataset: Pre-Trained Models in Open-Source Software. *PeaTMOSS* has three parts: a snapshot of (1) 281,638 PTMs, (2) 27,270 open-source software repositories that use PTMs, and (3) a mapping between PTMs and the projects that use them. We challenge *PeaTMOSS* miners to discover software engineering practices around PTMs. A demo and link to the full dataset are available at: https://github.com/PurdueDualityLab/PeaTMOSS-Demos.

I. High-Level Overview

Motivation: Deep Neural Networks (DNNs) have become a common component in software systems over the past decade. While some software engineers develop DNNs from scratch, many others integrate DNNs into their software following a typical re-use pattern: (1) pre-trained DNN models are published to registries such as Hugging Face, similar to traditional software package registries (*e.g.,* NPM, PyPI); and (2) other software depends on these Pre-Trained Models (PTMs), accessed via libraries or web APIs.

Despite wide-spread adoption of PTMs, we know relatively little about how PTMs are integrated into software systems.

Challenge: We propose the *PeaTMOSS* challenge to learn more about Pre-Trained Models in Open-Source Software (Figure 1).

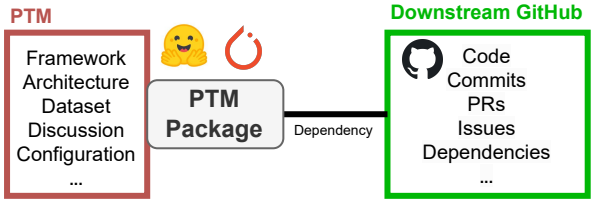

Fig. 1: Data for Pre-Trained Models in Open-Source Software.

The *PeaTMOSS* dataset comprises snapshots of PTMs and open-source repositories utilizing PTMs, as well as a mapping of PTMs to projects. *PeaTMOSS* contains 281,638 PTM packages, 27,270 GitHub projects that use PTMs as dependencies, and 44,337 links from these GitHub repositories to the PTMs they depend on. For both PTMs and GitHub projects, *PeaTMOSS* contains metadata (commits, issues, pull requests) and data (*e.g.,* model architecture and weights; `git` repositories). A uniform schema for retrieving PTM and project metadata is provided to assist in mining efforts. Most information is indexed; some is stored as blobs.

The dataset can be accessed in two formats. The "metadata" version of *PeaTMOSS* is 7.12 GB and contains only the metadata of the PTM packages and a subset of the GitHub project metadata. The "full" version is 48.2 TB, adding (1) the PTM package contents in each published version, and (2) `git` history of the main branches of the GitHub projects.

II. PeaTMOSS Dataset Structure

The metadata version of *PeaTMOSS* is stored in a SQLite database. The tables include hyperlinks to tarred copies of the PTM package or GitHub repository. Dataset schemas are described in §A.

The mapping between GitHub projects and PTMs are cases where a GitHub repository is known to depend on a particular PTM. Additional detail is given in §B-B.

III. Accessing and Working with PeaTMOSS

Access: The metadata and full versions of *PeaTMOSS* are available through a Globus share hosted at Purdue University. *PeaTMOSS* can be downloaded via the official Globus Connect application, which is available for all major operating systems, *e.g.,* Linux, MacOS, and Windows. We include a Python script to download and configure an SQLite instance of the metadata version. For more instructions, see https://github.com/PurdueDualityLab/PeaTMOSS-Demos.

Working with PeaTMOSS: To interact with *PeaTMOSS*, we recommend ORM or SQL. Examples are provided in §C.

Required Skills: *PeaTMOSS* is accessible to miners with a range of skills and interests. *PeaTMOSS* includes both standard mining artifacts from GitHub (*e.g.,* `git` repositories, issues, PRs) and unusual artifacts from PTMs (*e.g.,* neural networks, weights, model cards). Miners interested in program analysis, software repository mining, and natural language processing, etc. may apply these techniques to GitHub data, PTM data, or both.

*These authors contributed equally and are listed alphabetically.

Neither expertise in deep learning nor access to hardware such as GPUs is necessary for use of PeaTMOSS. Of course, miners with deep learning expertise or hardware resources could explore more advanced questions about PTM usage on GitHub or delve deeper into the PTM data.

Data Samples: We offer a subset of samples from *PeaTMOSS* at: https://github.com/PurdueDualityLab/PeaTMOSS-Demos.

IV. Possible Research Questions

Table I presents sample research questions for miners to investigate. This table includes questions focused on the GitHub portion of the dataset, on the PTM portion of the dataset, and on both parts. It notes some prior work as starting points for miners.

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

Research questions	Related work
GH1: 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?	[1]
GH2: What do developers on GitHub discuss related to PTM use, *e.g.,* in issues, and pull requests? What are developers' sentiments regarding PTM use? Do the people do pull requests of PTMs have the right expertise?	[2, 3]
GH3: How commonly do developers change the specific PTM they use to implement a feature? What factors influence these changes?	[4]
GH4: Sometimes a PTM is introduced into a GitHub repository as part of the initial implementation of a software feature, but other times a PTM is introduced to replace part or all of an existing feature implementation. How common are these two modes of PTM adoption? In the second kind, how does the feature's defect types or defect rates change after the PTM is introduced?	[5–8]
PTM1: What factors predict the popularity of a PTM, and what is their relative importance? Intuition suggests that performance aspects such as accuracy and latency may dominate; what is the role played by other factors such as engineering quality?	[9, 10]
PTM2: Recent qualitative work determined that software engineers struggle to re-use PTMs because of their limited documentation. What are the typical characteristics of this documentation? Can natural-language model cards be automatically parsed into a structured schema?	[10–12]
PTM3: One aspect of re-use is finding a candidate model. What naming conventions do PTMs follow? Are they consistent enough (within an architecture family? across families?) to support engineers looking for similar models? When do PTM maintainers release a model under a new name, and when do they simply bump the version number?	[13]
PTM4: PTM authors may reuse each others' work, *e.g.,* building off of model checkpoints or incorporating architectural building blocks. This might be viewed as an extreme form of "forking" from open-source software, but it may also reflect a novel form of software exchange. What is the phylogeny, or perhaps the "supply chain", of the major families of PTMs?	[11]
PTM5: Many research papers describe techniques for identifying DNNs with unexpected behavior, *e.g.,* hidden malicious behaviors. How common are such DNNs in the PTM dataset?	[10, 14–16]
I1: 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?	[17]
I2: What are the code smells related to PTM in the downstream projects, and how do they affect theses projects?	[18–21]
I3: When PTMs are used in GitHub repositories, what are engineers' testing practices for the PTMs they add as dependencies? Is there any correlation between the tests of the PTM by its maintainers, and the tests of the PTM by the downstream users? Do practices vary based on the purpose of the PTM, *e.g.,* computer vision vs. natural language processing? How do PTM downstream users deal with flakiness when testing a PTM?	[8, 22–24]
I4: Updating dependencies is a core software engineering activity. Suppose a GitHub repository depends on a PTM. How often does the GitHub repository update the dependency when that PTM is changed, *e.g.,* due to (1) PTM deprecation, (2) PTM improvement via a new version, or (3) PTM being made outdated by the release of a new model? What is the typical lag time for such updates?	[25, 26]
I5: Software engineers often communicate through filing issue reports. What are the characteristics of issue reports on the PTM packages, *e.g.,* in terms of the kinds of questions asked, responsiveness of maintainers, issue density, and issue staleness? How often does the topic of reproducibility come up (cf. the "ML reproducibility crisis")? How do these attributes differ from the characteristics of issue reports in GitHub repositories?	[27, 28]
I6: When engineers deploy software applications that make use of PTMs, they may prefer to use a deployment framework, *e.g.,* the ONNX RunTime, rather than a development framework such as PyTorch. Which of the several competing deployment frameworks (ONNX RunTime, MM-DNN, NNET, TFLite, etc.) is the most popular, and is there any evidence of why? Do GitHub users make the transformation to deployment themselves or do the PTM authors provide the deployment-ready version?	[29–31]

APPENDIX A
DATASET SCHEMA

The detailed schema is shown in Figure 2. The definition in SQL is available to miners.

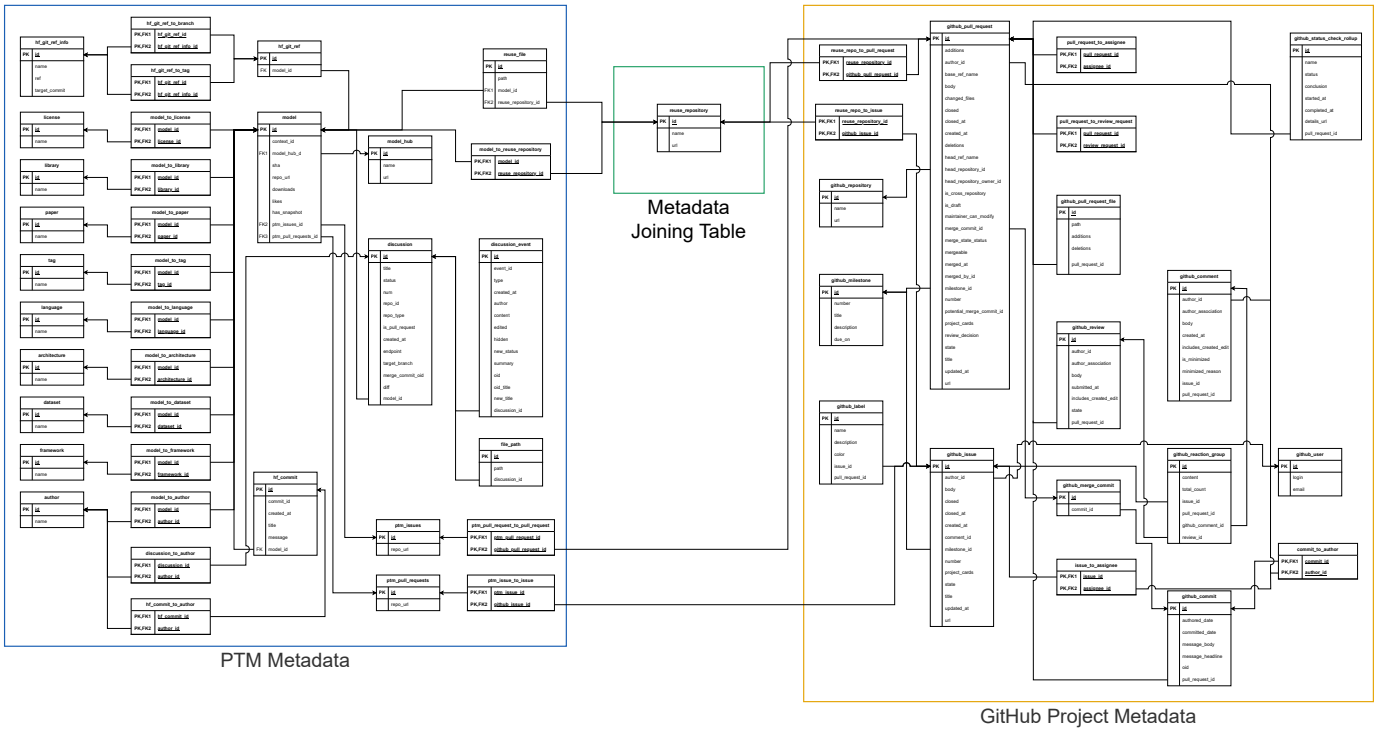

Fig. 2: The *PeaTMOSS* data schema. The database has three regions: one set of tables for PTMs, one set of tables for GitHub projects, and one table linking the two. The underlying PTMs and GitHub repositories are stored in the Globus share and can be fetched on demand. A more navigable version of the schema is available in the demo repository.

APPENDIX B
DATA COLLECTION

A. PTMs

1) What is a PTM and PTM package?

Pre-trained deep learning models (PTMs) are often shared through deep learning model registries, such as Hugging Face. Engineers can directly reuse these PTMs, or fine-tune them for specific downstream tasks. A PTM package typically includes a model card (a form of README), as well as the model's architecture and weights [10]. In recent years, the popularity of PTMs has been steadily rising [32, 33]. As illustrated in Figure 3, the total number of Hugging Face models has seen a consistent increase on a monthly basis. Recent work shows increasing interest from the software engineering mining community in PTMs [10, 11, 34, 35]. These works have identified the potential mining data that the community can take advantage of. In the past year the first mining efforts of PTMs and software engineering practices have begun [32, 33].

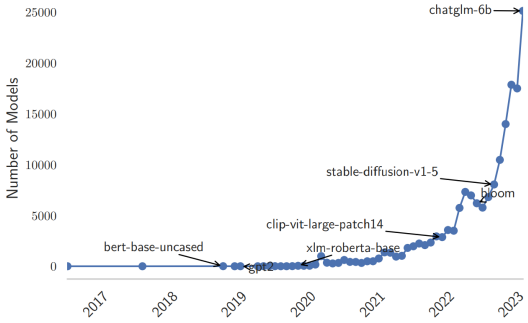

Fig. 3: Evolution of the total number of Hugging Face models per month. This figure is reused from [33].

2) PTM Collection

Our PTM data collection includes three parts: (1) We saved 15,250 PTM snapshots. This included the most popular PTM packages (*i.e.,* with over 10K downloads) on Hugging Face, which resemble a "git-like" structure, and all PTMs on PyTorch. 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 and allow miners to analyze the relationship between them. (3) For all PTMs hosted on Hugging Face and PyTorch, we retrieved their metadata, resulting in a total number of 281,638 PTM package metadata being included in *PeaTMOSS*. The miners can answer research questions based on metadata only, such as analyzing PTM naming conventions.

3) 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.

4) Implementation

Metadata is collected using an *Extract-Translate-Load (ETL)* pipeline for each model hub. The ETL pipeline can be generalized to the following steps:

- **Extract**: Obtain metadata that is available from each model hub's API.
- **Transform**: Use metadata to collect more information about PTMs (*i.e.,* examine Github Metadata for a linked repository) as well as download PTM package of GitHub repository snapshot. Transform the data into intermediate representation to simplify Loading.
- **Load**: Load the transformed data into a database for long-term storage and retrieval

Each model hub has a unique implementation of the Extract stage, but the functionality is the same.

- **Hugging Face**: PTM package metadata is downloaded using the `HuggingFace_hub` Python library.
- **PyTorch**: The markdown files in the root of the PyTorch GitHub repository, which correspond to each PTM package's repository, are downloaded and subsequently scraped for metadata.

During the Transform stage, data that matches the `PeaTMOSS.db` metadata schema is transformed into an intermediate representation, while data that doesn't match the schema is transformed into a JSONB blob. This is to allow for both consistency across model hubs, as well as maintaining hub specific metadata.

B. Mapping GitHub projects to PTMs

To evaluate PTM usage within projects, a PTM must be mapped back to a GitHub project. While it is possible to use Hugging Face and PyTorch Hub hosted projects with libraries outside of the Python ecosystem, we filtered on GitHub projects that utilize Python libraries to interface with PTMs as the vast majority of projects we found utilized Python libraries. As Hugging Face and PyTorch hub do not provide metadata or a method to retrieve PTM usage within projects, by analyzing the source code of a GitHub project, it is possible to map the two. The whole methodology has been described below:

1) Signature Collection

For leveraging Hugging Face PTMs, dedicated functions/methods specific to each library are available. These functions/methods allow loading PTMs by inputting their names as arguments. In this step, we manually retrieve the libraries, along with their corresponding import identifiers and the necessary function/method names essential for PTM loading. This compilation process is guided by Hugging Face's

official documentation.* The culmination of import identifiers and the essential function/method names is referred to as a Signature. For example, when accessing the PTMs provided by the Diffusers library, the corresponding signature encompasses `diffusers` and `from_pretrained`. Note that some libraries offer multiple methods to load PTMs, and we have taken all of these methods into account. For instance, the `transformers` library presents two distinct approaches: `from_pretrained` and `pipeline`, both of which can be utilized to load PTMs. Figure 4 shows an example code snippet to load the PTMs provided by `Transformers` library.

Throughout this process, we exclude libraries not visible on the website (such as `k2`, `doctr`, `mindspore`), as well as those unsuitable for downstream projects. An example of such incompatibility is when PTMs can only be used via command line or download, or when they lack the "use in library" feature†. After filtering, a total of 23 libraries remain in the compilation.

To use PyTorch PTMs, there are two approaches: (1) `torch.hub.load` function that provides a way to load PTMs by specifying their names as arguments. and (2) Alternatively, one can utilize library-specific classes provided by `torchvision` or instances provided by `torchaudio` and `torchtext`. These classes and instances are tailored to represent individual PTMs. The first approach and alternative approach provided by the `torchvision` library contain keyword based parameters i.e., pretrained/weights to control the model usage with pretrained weights or random initialization. The above mentioned two approaches give us a total of 163 signatures. Figure 5 shows an example code snippet to load the PTMs provided by the `PyTorch` library.

```
from transformers import AutoTokenizer, AutoModelForMaskedLM

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

Fig. 4: An Example Code Snippet to Use PTMs from Hugging Face Hub

```
import torch

# Model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')
```

Fig. 5: An Example Code Snippet to Use PTMs from PyTorch Hub

2) Preliminary repository collection based on Sourcegraph Search

The subsequent task involves locating these signatures within the code. Although we attempted to use GitHub search, it does not facilitate a comprehensive search as it provides the top 1000 results. Instead, we utilize the Sourcegraph command line interface, `src`‡, to detect projects containing pertinent files that employ the gathered signatures to interact with Hugging Face and PyTorch Hub.

Our search pattern incorporates the signatures gathered earlier, and we match them against the content of files within GitHub repositories. Our search criteria encompass repositories that are not archived (default behavior), not forked (default behavior), and publicly accessible, specifically focusing on Python files. For example, a query for `Diffusers` is structured as `"src select:file visibility:public count:all lang:Python content:'from diffusers' AND from_pretrained("`. Our search query accommodates both 'from import' and 'import' statements. Our search results include the corresponding code snippets, file names, and repository names. The projects were identified during July 5-12. Based on the count of the repositories, we select the top 5 Hugging Face libraries for our data collection including `Transformers`, `Spacy`, `Sentence-Transformers`, `Diffusers`, and `Timm`. For PyTorch, we consider all of the corresponding signatures. Our dataset comprises well-recognized GitHub repositories with an average star count of 201.

We have obtained local copies of the GitHub repositories by using the git clone command. This process took approximately 12 days to complete and resulted in the download of 36,251 repositories, collectively amounting to a total size of 3.5 TB.

3) Extracting PTMs via Static Analysis

As Sourcegraph's search feature relies on text-based patterns, the possibility of encountering false positive results exists. To mitigate this concern, we perform static analysis on GitHub repositories with the Scalpel framework [36]. 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 retrieve the complete and qualified names of each identified function call and cross-reference them with our predefined signatures which gives us a total of 28,575 repositories. Additionally, we go a step further by extracting both the positional and keyword arguments that are associated with the function calls that match our target signatures. Our analysis is equipped to capture any argument that possesses a static value. We then utilize the list of PTMs from PTM Torrent V2 to identify the repositories that statically call PTMs which gives us a total of 15,129 repositories. We store the corresponding repositories and files for each of the matched PTMs. It is important to note that a single repository can utilize multiple PTMs, and similarly, a single PTM can be employed across multiple repositories.

4) Soundness and Completeness of Collected Repositories

For the PTMs hosted on the Hugging Face hub, our dataset provides usage considering the five libraries, i.e., `Transformers`, `Spacy`, `Timm`, `Sentence-Transformers`, and `Diffusers`. These libraries were chosen because they comprise the top five libraries used in GitHub projects as shown in Figure 6. For the PTMs from the PyTorch hub, we did not filter by the library. Our dataset comprises `torchvision`, `torchaudio`, `torchtext`, along with PyTorch hub.

*https://github.com/HuggingFace/hub-docs/blob/main/js/src/libVinterfaces/Libraries.ts

†https://HuggingFace.co/facebook/musicgen-large

‡https://docs.sourcegraph.com/cli

TABLE II: JSON response fields captured when collecting issues and pull requests for 28,575 GitHub repositories.

Request Type	Response Fields
Issue Metadata	assignees, author, body, closed, closedAt, comments, createdAt, id, labels, milestone, number, projectCards, reactionGroups, state, title, updatedAt, url
Pull Request Metadata	additions, assignees, author, baseRefName, body, changedFiles, closed, closedAt, comments, commits, createdAt, deletions, files, headRefName, headRepository, headRepositoryOwner, id, isCrossRepository, isDraft, labels, maintainerCanModify, mergeCommit, mergeStateStatus, mergeable, mergedAt, mergedBy, milestone, number, potentialMergeCommit, projectCards, reactionGroups, reviewDecision, reviewRequests, reviews, state, statusCheckRollup, title, updatedAt, url

Static analysis was carried out due to the limitations of the text search conducted using Sourcegraph. We resolve the fully qualified names for each function call to accurately identify True Positive results. This results in a total of 28,575 repositories that genuinely contain the practical utilization of the PTMs. Our dataset encompasses projects created up until July 10, 2023.

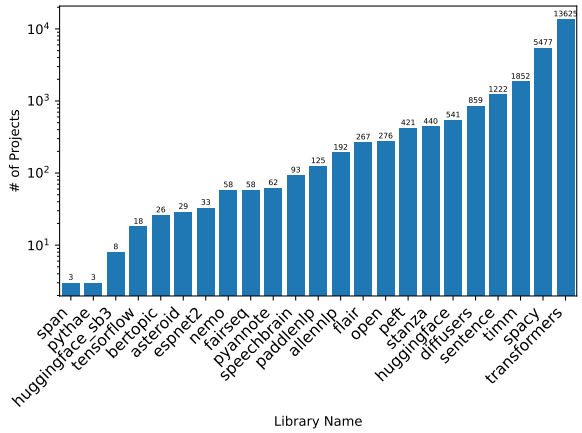

Fig. 6: Number of projects that use a specific library as captured via Sourcegraph code search

5) Extracting GitHub Issues and Pull Requests

By analyzing the discussions that community members have about the project within the project's issue tracker, it is possible to identify not only PTMs of interest w.r.t the project, but the potential future direction of a project w.r.t the techniques implemented by the PTM.

To collect the issues and pull requests associated with the GitHub repositories, we use GitHub's official command line interface `gh`. We consider all states (*i.e.,* open and closed) while collecting the issues and pull requests associated with each repository. Each of the issue and pull request metadata responses contain all available relevant fields provided by GitHub CLI. Specific response fields are listed in Table II. Example commands to retrieve relevant data can be found in *PeaTMOSS* GitHub repository. Targeting 28,575 repositories, we retrieve the issues or pull requests resulting in 19,507 repositories with issues and 12,159 repositories with pull requests.

Altogether, our dataset encompasses a total of 27,270 repositories, which involve occurrences of issues, pull requests or static utilization of PTMs.

APPENDIX C
DATA ACCESS EXAMPLES

To answer several of the proposed research questions in Table I, we have released examples on how to interface with *PeaTMOSS*. ORM methods and SQL examples for interfacing with the `PeaTMOSS.db` database are provided. Code snippets for these examples are made available via the `/Examples/` filepath in our GitHub repository.

REFERENCES

[1] M. M. Morovati, A. Nikanjam, F. Tambon, F. Khomh, and Z. Ming, "Bug characterization in machine learning-based systems," *arXiv*, 2023. [Online]. Available: https://arxiv.org/abs/2307.14512

[2] L. Yin and V. Filkov, "Team discussions and dynamics during devops tool adoptions in oss projects," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 697–708.

[3] A. Sajadi, K. Damevski, and P. Chatterjee, "Interpersonal trust in oss: Exploring dimensions of trust in github pull requests," in *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2023, pp. 19–24.

[4] 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 (TOSEM)*, vol. 30, no. 4, pp. 1–42, 2021.

[5] Y. Li, Z. Zhang, B. Liu, Z. Yang, and Y. Liu, "Modeldiff: Testing-based dnn similarity comparison for model reuse detection," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 139–151.

[6] Z. Zhang, Y. Li, J. Wang, B. Liu, D. Li, Y. Guo, X. Chen, and Y. Liu, "Remos: reducing defect inheritance in transfer learning via relevant model slicing," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1856–1868.

[7] B. Qi, H. Sun, X. Gao, H. Zhang, Z. Li, and X. Liu, "Reusing deep neural network models through model re-engineering," *arXiv preprint arXiv:2304.00245*, 2023.

[8] N. Nahar, H. Zhang, G. Lewis, S. Zhou, and C. Kästner, "A dataset and analysis of open-source machine learning products," *arXiv preprint arXiv:2308.04328*, 2023.

[9] C. Lima and A. Hora, "What are the characteristics of popular apis? a large-scale study on java, android, and 165 libraries," *Software Quality Journal*, vol. 28, no. 2, pp. 425–458, 2020.

[10] 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 *IEEE/ACM 45th International Conference on Software Engineering (ICSE'23)*, 2023.

[11] W. Jiang, N. Synovic, and R. Sethi, "An Empirical Study of Artifacts and Security Risks in the Pre-trained Model Supply Chain," *Los Angeles*, p. 10, 2022.

[12] A. Bhat, A. Coursey, G. Hu, S. Li, N. Nahar, S. Zhou, C. Kästner, and J. L. Guo, "Aspirations and practice of ml model documentation: Moving the needle with nudging and traceability," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–17.

[13] R. Gresta, V. Durelli, and E. Cirilo, "Naming practices in object-oriented programming: An empirical study," *Journal of Software Engineering Research and Development*, pp. 5–1, 2023.

[14] S. Wang, S. Nepal, C. Rudolph, M. Grobler, S. Chen, and T. Chen, "Backdoor Attacks Against Transfer Learning With Pre-Trained Deep Learning Models," *IEEE Transactions on Services Computing*, vol. 15, no. 3, pp. 1526–1539, May 2022.

[15] Z. Wang, C. Liu, X. Cui, J. Yin, and X. Wang, "EvilModel 2.0: Bringing Neural Network Models into Malware Attacks," *Computers & Security*, 2022.

[16] S. Guo, C. Xie, J. Li, L. Lyu, and T. Zhang, "Threats to pre-trained language models: Survey and taxonomy," *arXiv preprint arXiv:2202.06862*, 2022.

[17] Y. Fan, X. Xia, D. Lo, A. E. Hassan, and S. Li, "What makes a popular academic ai repository?" *Empirical Software Engineering*, vol. 26, pp. 1–35, 2021.

[18] F. Palomba, D. A. Tamburri, F. A. Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE transactions on software engineering*, vol. 47, no. 1, pp. 108–129, 2018.

[19] H. Zhang, L. Cruz, and A. Van Deursen, "Code smells for machine learning applications," in *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, 2022, pp. 217–228.

[20] B. Van Oort, L. Cruz, M. Aniche, and A. Van Deursen, "The prevalence of code smells in machine learning projects," in *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 2021, pp. 1–8.

[21] N. Cardozo, I. Dusparic, and C. Cabrera, "Prevalence of code smells in reinforcement learning projects," *arXiv preprint arXiv:2303.10236*, 2023.

[22] S. Li, J. Guo, J.-G. Lou, M. Fan, T. Liu‡, and D. Zhang, "Testing Machine Learning Systems in Industry: An Empirical Study," in *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2022, pp. 263–272.

[23] H. B. Braiek and F. Khomh, "On testing machine learning programs," *Journal of Systems and Software (JSS)*, vol. 164, p. 110542, 2020.

[24] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli, "Understanding flaky tests: The developer's perspective," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 830–840.

[25] A. Hora, R. Robbes, M. T. Valente, N. Anquetil, A. Etien, and S. Ducasse, "How do developers react to api evolution? a large-scale empirical study," *Software Quality Journal*, vol. 26, pp. 161–191, 2018.

[26] C. Wan, S. Liu, H. Hoffmann, M. Maire, and S. Lu, "Are machine learning cloud apis used correctly?" in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 125–137.

[27] W. Jiang, V. Banna, N. Vivek, A. Goel, N. Synovic, G. K. Thiruvathukal, and J. C. Davis, "Challenges and practices of deep learning model reengineering: A case study on computer vision," *arXiv preprint arXiv:2303.07476*, 2023.

[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," *arXiv preprint arXiv:2303.09795*, 2023.

[29] M. Liu, C. Zhao, X. Peng, S. Yu, H. Wang, and C. Sha, "Task-oriented ml/dl library recommendation based on a knowledge graph," *IEEE Transactions on Software Engineering*, pp. 1–16, 2023.

[30] Z. Zhang, L. K. Ng, B. Liu, Y. Cai, D. Li, Y. Guo, and X. Chen, "Teeslice: slicing dnn models for secure and efficient deployment," in *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, 2022, pp. 1–8.

[31] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán, and Á. Rodríguez-Vázquez, "Optimum selection of dnn model and framework for edge inference," *IEEE Access*, vol. 6, pp. 51 680–51 692, 2018.

[32] A. Kathikar, A. Nair, B. Lazarine, A. Sachdeva, and S. Samtani, "Assessing the vulnerabilities of the open-source artificial intelligence (ai) landscape: A large-scale analysis of the hugging face platform," 2023.

[33] J. Castaño, S. Martínez-Fernández, X. Franch, and J. Bogner, "Exploring the carbon footprint of hugging face's ml models: A repository mining study," *arXiv*, 2023. [Online]. Available: https://arxiv.org/pdf/2305.11164.pdf

[34] 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 IEEE John Vincent Atanasoff Symposium on Modern Computing (JVA'23)*, 2023.

[35] A. Ait, J. L. C. Izquierdo, and J. Cabot, "Hfcommunity: A tool to analyze the hugging face hub community," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2023, pp. 728–732.

[36] L. Li, J. Wang, and H. Quan, "Scalpel: The python static analysis framework," *arXiv preprint arXiv:2202.11840*, 2022.