Naming Practices of Pre-Trained Models in Hugging Face

WENXIN JIANG, Purdue University, West Lafayette, IN, USA
CHINGWO CHEUNG, Purdue University, West Lafayette, IN, USA
MINGYU KIM, Purdue University, West Lafayette, IN, USA
HEESOO KIM, Purdue University, West Lafayette, IN, USA
GEORGE K. THIRUVATHUKAL, Loyola University Chicago, Chicago, IL, USA
JAMES C. DAVIS, Purdue University, West Lafayette, IN, USA

As innovation in deep learning continues, many engineers seek to adopt Pre-Trained Models (PTMs) as components in computer systems. Researchers publish PTMs, which engineers adapt for quality or performance prior to deployment. PTM authors should choose appropriate names for their PTMs, which would facilitate model discovery and reuse. However, prior research has reported that model names are not always well chosen — and are sometimes erroneous. The naming for PTM packages has not been systematically studied.

In this paper, we frame and conduct the first empirical investigation of PTM naming practices in the Hugging Face PTM registry. We initiated our study with a survey of 108 Hugging Face users to understand the practices in PTM naming. From our survey analysis, we highlight discrepancies from traditional software package naming, and present findings on naming practices. Our findings indicate there is a great mismatch between engineers' preferences and practical practices of PTM naming. We also present practices on detecting naming anomalies and introduce a novel automated *D*NN *AR*chitecture *A*ssessment technique (DARA), capable of detecting PTM naming anomalies. We envision future works on leveraging meta-features of PTMs to improve model reuse and trustworthiness.

ACM Reference Format:
Wenxin Jiang, Chingwo Cheung, Mingyu Kim, Heesoo Kim, George K. Thiruvathukal, and James C. Davis. 2024. Naming Practices of Pre-Trained Models in Hugging Face. 1, 1 (March 2024), 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

A Pre-Trained Model (PTM) packages the capabilities of a deep neural network (DNN) trained on a task. Engineers exchange PTMs on registries (*e.g.,* Hugging Face) similar to how traditional software components are shared (*e.g.,* NPM, PyPI, or Maven packages). The selection of traditional software components depends on the availability of ecosystems, knowledge base, and examples [1, 2]. Somewhat differently from traditional software components [3, 4], software engineers use the *names* of PTMs during PTM reuse. Specifically, they search for DNN architectures by name, and select among PTMs based on the training regime, architecture, dataset, etc., expressed in the name.

Although software engineers make use of PTM names during model search and selection, these processes are challenging [5, 6]. In their interview data, Jiang *et al.* reported that engineers use PTM

Authors' addresses: Wenxin Jiang, Purdue University, West Lafayette, IN, USA, jiang784@purdue.edu; Chingwo Cheung, Purdue University, West Lafayette, IN, USA, cheung59@purdue.edu; Mingyu Kim, Purdue University, West Lafayette, IN, USA, kim3118@purdue.edu; Heesoo Kim, Purdue University, West Lafayette, IN, USA, kim2903@purdue.edu; George K. Thiruvathukal, Loyola University Chicago, Chicago, IL, USA, gkt@cs.luc.edu; James C. Davis, Purdue University, West Lafayette, IN, USA, davisjam@purdue.edu.

XXXX-XXXX/2024/3-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

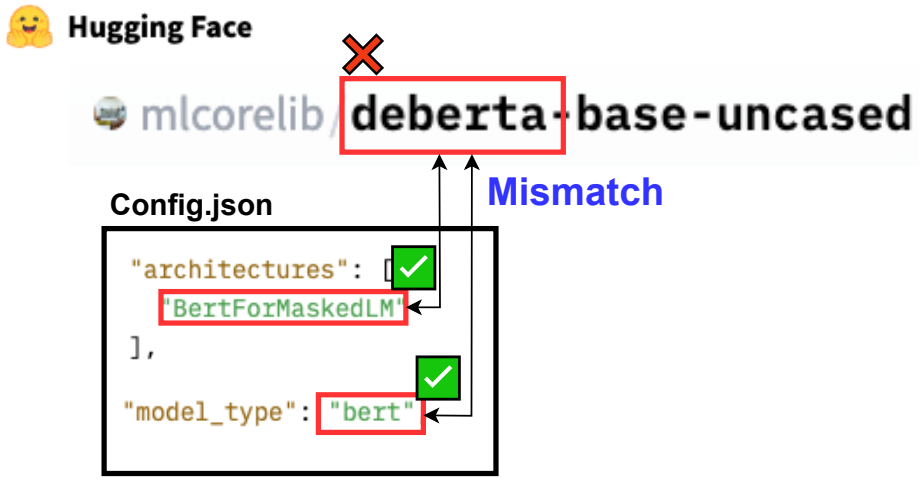


Fig. 1. Example of a PTM naming anomaly from Hugging Face. Top part: PTM identifier. Bottom part: PTM architecture and model types. The identifier and architecture do not match.

names in their reuse process but that these names are viewed as flawed (Figure 1) or difficult to comprehend, thus hindering the efficiency of PTM search and selection [4]. Industry leaders have also emphasized the importance of PTM metadata (*e.g.,* PTM names) in model registries, because this facilitates better PTM management [7, 8]. As argued by Davis *et al.* [5], a comprehensive understanding of the naming convention within the PTM ecosystem would benefit PTM authors (improved discoverability); PTM reusers (improved search and selection); and registry operators (better user experience).

Therefore, the goal of this work is to delineate the naming conventions of PTMs. We focus on two themes: naming practices and rationales, and naming anomalies. In the first theme, our objective is to comprehend the prevalent naming practices, identify the inherent challenges, and discern how PTM naming diverges from traditional software package naming conventions. We take a mixed-methods approach here. We conducted a survey involving 108 Hugging Face users, achieving a 95% confidence level with a margin of error of 10%. We triangulate that data with a mining study of real-world PTM names, analyzing 14,296 PTMs (1.7% of Hugging Face PTMs) from the PeaTMOSS dataset [9]). For the second theme, we designed and evaluated an automated tool to detect architecture-related naming anomalies.

Our findings indicate that PTM naming is different from traditional software package naming. Engineers follow specific naming practices and encounter challenges that are specific to PTM naming. Based on our survey data, we summarize the differences between traditional package naming and PTM naming, and present the engineers' perspective of PTM naming practices. Our empirical measurement indicates the practical distribution of naming elements are different from engineers' preferences. Our automated tool, DARA, shows its capability of detecting naming anomalies.

Our contributions are:

- We delineate the PTM naming practices by conducting a survey study. We delineated the PTM naming practices by conducting a comprehensive survey among 108 Hugging Face users. This survey helped define the nature of naming practices in the PTM ecosystem. (§5-RQ1, 2, 3))

- We triangulated the survey study with measurements of 14,296 PTM names from Hugging Face. We developed a GPT-4 based technique to automatically extract the naming elements from PTM names. (§5-RQ2, 4)
- We developed a novel algorithm, DNN Architecture Assessment (DARA), to detect naming anomalies. (§5.2)
- We identify unique properties of PTM naming to guide future research on improved automated tool development, and enhancing both model reusability and trustworthiness. (§7)

Significance: Naming practices and conventions are critical in the context of pre-trained models (PTMs). We conducted the first empirical measurements on PTM naming conventions through a survey study and repository mining study. We also contributed automated measuring tools focusing on measuring naming conventions in Hugging Face and automated discovery of naming anomalies. Our findings help engineers and researchers have a better understanding of the PTM naming practices and provide an initial solution to this problem domain. Our findings offer PTM creators and maintainers insights to enhance the quality of PTM packages, and highlight how software engineering tools can be leveraged to advance further naming analysis.

2 Background and Related Work

This section includes the background and related work about software component naming (§2.1), how PTMs are named in registries (§2.2), and the challenges of PTM reuse (§2.3).

2.1 Naming Components in Software Engineering

Component-based development offers a systematic approach to constructing software systems from reusable parts, enabling increased reuse and reduced software production cost [3, 10]. By leveraging existing software components, developers can build upon established, tested functionalities instead of creating solutions from scratch [11]. This enhances efficiency and promotes consistency and reliability across software projects.

Searching and selecting software components for reuse is easier when the components have good names. For example, in a traditional software ecosystem (not focused on AI/ML) such as PyPI, it is clearer that package `matplotlib` is about plotting than is its competitor package `seaborn`, which could facilitate its discovery and subsequent selection.

The naming of software components is also crucial for code readability, maintainability, and collaboration [12–14]. Prior works indicated that software components are normally described in "how it does it" (*i.e.,* implementation units), but some components can also be described in "what a components does" (*i.e.,* application goals) [15, 16]. Existing studies have primarily concentrated on the naming of functions [17], classes [18], and variables [16], neglecting the naming conventions of packages. In the context of deep learning, PTM packages represent critical software components. This work sheds light on that specific to PTM packages.

2.2 How PTMs are Named in Model Registries

In this section, we provide an overview of the naming framework used in model registries based on the official documentation, particularly Hugging Face [19].

In the rapidly evolving landscape of PTMs, naming conventions are vital for ensuring both consistent communication among stakeholders and compatibility across different models [4]. Existing literature notes that the prevalent approach to PTM reuse is conceptually fork-based, incorporating both architectural modifications and additional training [20, 21].

PTM Package Naming Many model hubs have been developed to ease the reuse of PTMs. These hubs generally adopt similar naming practices, which typically include the problem domain, task, model identifier, and related files [22]. Previous research has defined and examined current

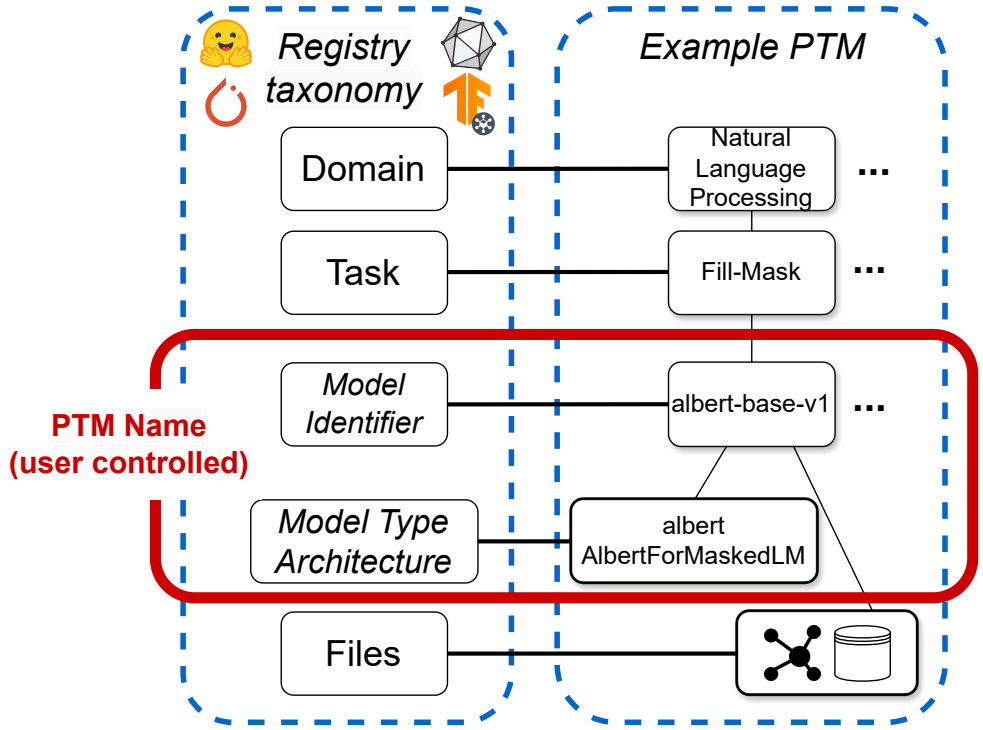


Fig. 2. Hierarchy of PTM registry naming taxonomy. The red box highlights the package name (model identifier and model type/architecture) — this is the focus of our study.

PTM registries, revealing that Hugging Face is the most popular due to its open-access approach, allowing anyone to contribute to and reuse PTM packages [23]. Hugging Face employs a multi-tiered categorization system, spanning diverse domains, tasks, PTMs, and configurations [19]: A *domain* is the general field or area of application where a set of models operate, such as Natural Language Processing or Computer Vision. Within a domain, *tasks* are specific problems that models aim to solve, like Sentiment Analysis or Named Entity Recognition in the Natural Language Understanding domain. A *PTM* is an individual instance of a trained model for a specific task, complete with weights, parameters, and configuration metadata. A PTM package's `config.json` file details the model's *architecture* and specifies the *model type* and the task it is designed for, *e.g.,* `Albert` for the model type and `MaskedLM` for the task.

Parts of this naming taxonomy (Figure 2) are determined by model registries, such as Hugging Face, and parts are determined by authors:

- Domain and task are high-level metadata that are part of the *package type*.
- We define a ***PTM name*** as the combination of a package *identifier* (*e.g.,* `albert-base-v2`) and the *model type* or *architecture* indicated in the metadata (*e.g.,* `AlbertForMaskedLM`).
- *Package content*: low-level files included in a package, such as weights and training scripts.

In the Hugging Face ecosystem, both the package type and its content are objective factors. On the other hand, the package name is a subjective element, often produced manually by the PTM contributors themselves. This subjectivity makes the naming process a critical point where inadequacies or ambiguities can occur. It is essential to recognize that both objective and subjective components of a package are susceptible to varying degrees of judgement, which can impact

how reengineers and adopters search for models. According to prior research [4], the engineers involved in PTM ecosystem commonly rely on either package identifiers or architectural labels when searching for PTM packages appropriate for their specific tasks.

Naming Elements In the context of PTM package naming, naming elements comprising a model's name, representing different categories or attributes that are combined to form the full model name. For example, a model with elements of "Architecture-Size-Dataset Characteristic," indicating that the model name is composed of information about its architecture (`e.g., deberta`), size (*e.g.,* `base`), and dataset characteristic (*e.g.,* `uncased`).

2.3 The Challenges of PTM Reuse

PTM reuse has garnered increasing attention in the academic community. Previous studies have focused on various aspects of PTM reuse, including environmental impacts [24], vulnerabilities [25], and lifecycle management [26]. Taraghi *et al.* conducted a mining study and indicated that the most challenging part of PTM reuse is model usage and understanding [27]. Addressing the intricacies of PTM naming conventions is critical, as clear and consistent naming can significantly facilitate engineers' comprehension of package contents and contribute to reducing production costs [12, 28]. However, standardized PTM naming practices have yet to be established. Our work provides the first empirical measurement on the naming practices of PTM packages in Hugging Face.

According to Jiang *et al.*, PTM reuse is distinct from traditional software reuse process and has many challenges, including naming ambiguity, absence of comprehensive documentation, and compatibility issues [4, 9]. Some of the 12 interview participants from their study mentioned the challenges of PTM component reuse specifically. There are 6 out of 12 interview participants in their study mentioned the untrustworthiness of model architectures, which is an important components of PTM naming. Two participants specifically mentioned the reuse challenges caused by the naming practices of PTM packages [4].

Naming Anomalies *Naming anomalies* signify inconsistencies or elements not aligned with standard practices. These anomalies can lead to misunderstandings or mistakes in reuse. For instance, labeling "`ResNet`" as "`VGGNet`" would be a naming anomaly beyond mere subjectivity. According to a prior interview study [4], engineers often rely on the names of PTM packages for information, as model cards (README) and metadata provided by these registries are often insufficient. Such missing details and inconsistencies can pose challenges in PTM reuse, including problems with understanding the model's capabilities and limitations, as well as issues related to code maintenance and collaboration [14].

Figure 1 shows an example PTM naming anomaly. It is important to note that the appropriateness of a PTM name can be subjective. However, there is no standard or guidelines exist to assess this in the PTM ecosystem. Our study illustrates the methods practitioners employ to identify such anomalies and introduces an automated tool designed to detect incorrect architecture and model types.

3 Research Questions

To summarize the literature: previous studies have focused on the naming conventions associated with traditional software packages [13, 14]. Existing work has shown how PTM reuse is challenging and the importance of systematic PTM naming [4, 5]. PTM naming presents its own set of challenges that have yet to be discovered.

To address this gap and identify the potential risks of naming in the PTM ecosystem, our study includes two themes: (1) measurements of naming practices and (2) discovery of naming anomalies. We ask:

Theme 1: Measurements of Naming Practices

RQ1 How is PTM naming different from traditional software package naming?
RQ2 What elements should be included in a PTM identifier?

Theme 2: Discovery of naming anomalies

RQ3 How do engineers identify naming anomalies?
RQ4 How to automatically identify architectural anomalies?

Theme 1 focuses on understanding the naming practices of PTM packages in Hugging Face. Theme 2 focuses on practical practices and a novel approach to automatically discover naming anomalies.

4 Methodology

This section presents our methodology of a mixed-method approach to answer RQ1-3 [29]. We conducted a survey study, then we substantiate with practical measurements in Hugging Face. Details of RQ4 will be presented in §5.2. Figure 3 shows the relationship between RQs and methods.

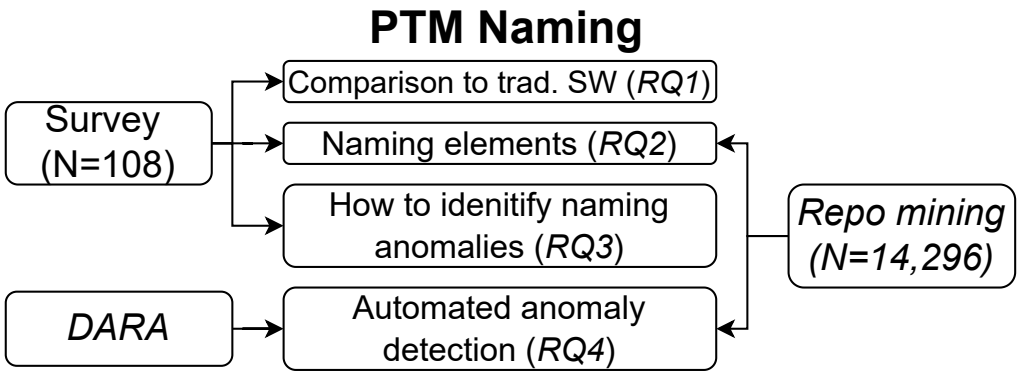


Fig. 3. Relationship between RQs and research methodology.

4.1 Survey Study

This section presents our methods of survey study, including preliminary analysis (§4.1.1), instrument design (§4.1.2), recruitment (§4.1.3), and analysis (§4.1.4).

4.1.1 Preliminary Analysis We conducted a preliminary analysis on 200 models to understand the naming conventions of PTMs packages. Based on those results, we identified 12 naming elements and 4 naming conventions (Table 4). We expanded our preliminary analysis to an additional 50 models and observed no emergence of new categories. Consequently, we determined that our identified naming conventions and elements are comprehensive. The corresponding definitions and examples are in Table 1.

4.1.2 Survey Instrument Design We developed our survey instrument following Kitchenham's guidelines for survey design [30]. We began with an extensive review of literature about software naming [12–14] and PTM reuse [4, 23], shaping our initial survey instrument. The initial instrument includes five sections: demographic questions, comparison between PTM naming and traditional software engineering, PTM naming elements and conventions, and practices on naming anomaly detection.

To improve the survey, we conducted two pilot studies. Firstly, we refined the initial version through a pilot study with 3 subjects, aiming to enhance the clarity and logical flow based on participant feedback. The pilot study took 15–20 minutes. Secondly, we conducted another pilot

Table 1. Definition and examples of PTM naming components. Others were identified based on the results of our pilot study.

Naming component	Definition	Example
Architecture [A]	The foundational structure or design of the deep learning model.	bert, albert, resnet
Model size [S]	The model complexity *e.g.,* # of parameters or architectural units.	50, 101, base, large, xxlarge
Dataset [D]	The specific dataset used for training or evaluation of the model.	squad, imagenet
Dataset characteristics [C]	The traits of the dataset, *e.g.,* text casing, image dimensions.	case, uncased, 1024-1024, 244
Model versioning [V]	The version of the model.	v1, v2
Reuse method [F]	The approach employed to adapt a PTM for downstream tasks.	fine-tune, distill, few-shot
Language [L]	The human language for which the model is trained on.	english, chinese, arabic
Task [T]	The problem or functionality the model is designed to address.	qa, cls, face-recognition
Training process [R]	The algorithm or approach used for training the model.	pretrain, sparse
Number of layers [Y]	The depth of the neural network in terms of layers.	L-12, L-24
Number of parameters [P]	The total number of trainable parameters in the model.	100M, 8B
Other [O]	A portion of the model name not classified above categories.	demo, test

study to evaluate if we could get valuable data. We distributed the survey to 30 Hugging Face PRO users and 100 ORG users in this pilot phase. However, the response rate was very low (1%). In response, we consolidated numerous questions, particularly short-answer types, and restructured the naming sections. Retesting with the initial 3 pilot participants, the revised survey took 5-8 minutes to complete.

The final survey in this study includes demographic questions and PTM naming practices. All the questions in the survey are optional to answer. We also included the definition of PTM naming (§2.2) as part of the survey. The final instrument is available in **??**. Our study was approved by institutional IRB.

4.1.3 Recruitment We compiled email addresses from Hugging Face's PRO and ORG (*i.e.,* organization) user accounts. PRO accounts mean that the users actively pay for the features provided by Hugging Face. To filter for practitioners, we collected users from organizations which were labeled as `company`, `community`, and `non-profit`, excluding `university`, `classroom`, and no labeled organizations.

Our survey aimed to achieve a 95% confidence level with a margin of error of 10. Previous statistics indicate that Hugging Face had over 1.2 million users by 2022 [31], and it kept increasing in recent years. Therefore, we needed at least 96 participants to ensure statistical significance [32].

The prior statistic shows that Hugging Face has at least 1.2 million users as of 2022 [31]. For this population, we require a minimum of 96 participants to achieve a confidence level of 95% with a 10% margin of error in our study.

Hence, we sent out surveys in batches of 100 emails until reaching our desired sample size. In total, we distributed our survey to 228 PRO users and 1,985 organization members. As an incentive, each participant was compensated with a $10 gift card. 119 users responded to our survey (108 actively interacted with Hugging Face), resulting in a response rate of 5.4%. Table 2 gives subject demographic information.

Table 2. Participant demographics (N=108 but some left blank.)

Type	Break-down
ML Exp.	<1 yr. (11), 1-2 yr. (24), 3-5 yr. (32), >5 yr. (36)
SE Exp.	<1 yr. (6), 1-2 yr. (12), 3-5 yr. (22), >5 yr. (63)
Org. Size	Small (1-50 employees, 58), Medium (51-250 employees, 19), Large (over 250 employees, 26)
Deployment Env.	Web application (68), Desktop application (21), Cloud and data center (61), IoT/embedded systems (12), Mobile (15), Other (6)
#used PTMs	0 (0), 1-5 (38), 6-10 (18), 11-20 (19), > 20 (28)
#created PTMs	0 (34), 1-5 (37), 6-10 (15), 11-20 (5), > 20 (12)

4.1.4 Survey Analysis The majority of our survey questions were formulated as multiple-choice/checkbox, which are simple to analyze. Two open-ended textual responses required subjective analysis.

The first open-ended question, for RQ1, asked: *"How is PTM naming similar/different from naming traditional software packages such as those on NPM or PyPI? Why do you think that is?"*. Responses were lengthy and variable, so two analysts worked together to analyze them following thematic coding methods [33]. First, the analysts independently performed memoing [34] to develop initial themes. Next, they met to iteratively refine the set of themes (initial codebook). Then, they applied the themes and assessed their reliability. The codebook was refined to merge some entangled codes and discard rare ones. They did a second round of coding and then met to reach agreement on disputes. They initially agreed on ~65% of codes. On the rest, reaching agreement typically added or subtracted one code from their joint set.

The second open-ended question, for RQ4, asked respondents to describe their practices for identifying naming anomalies. The analysis was thus handled by a single researcher.

4.2 Practical Measurement in Hugging Face

4.2.1 Data Collection We use the latest PTM dataset named PeaTMOSS [9], which was collected in August 2023. In our study, we focused on the identifiers of 14,296 PTM packages from PeaTMOSS dataset which has over 50 downloads.

4.2.2 LLM Pipeline Design Large language models have demonstrated significant capabilities in natural language understanding [35]. We developed a large-language model (LLM) pipeline using the ChatGPT API (`gpt-4-turbo`) to measure the PTM naming practices.

Figure 4 illustrates the LLM pipeline we developed. In this process, we input the name of a PTM package and employ two distinct prompts (§4.2.3, §4.2.3) to deduce the naming conventions and elements associated with that name.

Following the methodology suggested by Zamfirescu *et al.* [36], we utilized a structured process to design the prompt for our measurements. For each measurement, we first crafted a textual input (*i.e.,* prompt) and conducted a preliminary analysis on 20 randomly selected models. This process underwent iterative refinement to achieve two primary objectives: (1) Ensuring the generated output adheres to a well-defined, easily parsable format; (2) Optimizing the performance of the output, evaluated through manual inspection.

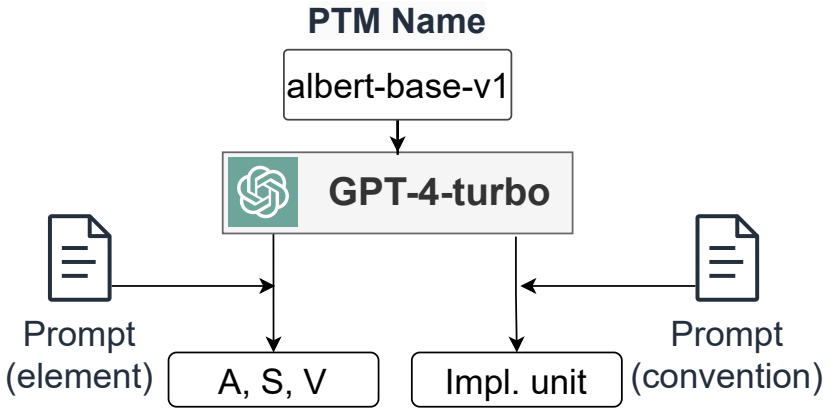


Fig. 4. Example of input/output of GPT-4. Input is model names. Output is semantic patterns and confidence level.

Initially, we incorporated element definitions from our survey into the prompt to clarify naming conventions. We then enrich the prompt with definitions and examples for each specified component to unify semantic naming patterns. We added several rules regulate the output format, and apply few-shot learning, presenting example inputs alongside desired outputs. We report our evaluation results in §4.2.3 and §4.2.3.

4.2.3 Evaluation In this section, we present the evaluation method and results of our two practical measurements.

Measurement of Naming Elements We randomly sampled 300 models from the model list. This gave a confidence level of 95%. One researcher manually labeled the semantic patterns of their names. We compared our manual labels to the output from the pipeline, and calculated the F1-score (91%). This indicate that our results have our results from the pipeline are reliable.

Table 3. Evaluation of naming element measurements. We evaluated the results from 300 randomly selected names.

Element	Precision	Recall	F1-Score
Architecture [A]	0.98	0.98	0.98
Model size [S]	0.96	1.00	0.98
Dataset [D]	0.94	0.89	0.92
Dataset characteristics [C]	1.00	0.74	0.85
Model versioning [V]	1.00	0.94	0.97
Reuse method [F]	1.00	0.87	0.93
Language [L]	0.95	0.92	0.94
Task [T]	0.90	0.91	0.91
Training process [R]	0.80	0.80	0.80
Number of layers [Y]	0.67	1.00	0.80
Number of params. [P]	1.00	0.96	0.98
Other [O]	0.87	0.81	0.84
Average	0.92	0.90	0.91

Measurement of Naming Conventions To assess the efficacy of this measurement approach, we initially established a ground-truth dataset. Following the definition in the survey, two researchers independently annotated 200 PTM package names, resulting in an initial Kappa coefficient of 0.72 (substantial). After achieving consensus on the labels through collaboration, these agreed-upon labels were established as the ground-truth for our analysis. We utilized this refined dataset to assess the accuracy of our LLM pipeline. The detailed evaluation results are presented in Table 4.

Table 4. Evaluation of naming convention measurements. We evaluated the results of 50 randomly selected names.

Naming Convention	Precision	Recall	F1-Score
Impl. unit	0.79	0.94	0.86
App. or Task	1.00	1.00	1.00
Impl. + App./Task	0.92	0.73	0.81
Other	1.00	1.00	1.00
Average	0.90	0.89	0.89

5 Results and Analysis

RQ1: How is PTM naming different from traditional software package naming?

Finding 1: PTM naming is different from traditional naming.

88% of respondents perceive PTM naming to be different from traditional package naming. The main difference is the greater semantic knowledge embedded in PTM names. The main explanations for this difference were: (1) differences in evolution practices; and (2) different information needed for reuse decisions.

A total of 52 respondents replied to the relevant survey (§4.1.4). We focused on the common case (88%): responses describing differences. We analyzed responses in two ways: *how* the names differ, and *why* the names differ.

Table 5 summarizes the codes and response frequency related to *how* names differ between PTMs and traditional software packages. The primary difference was in the greater amount of semantic knowledge in PTM names, as depicted in Table 1. A representative answer was "*Pre-trained model names often include more specific information about the model's architecture, training data, and performance. This...detail is not typically included in traditional...package names.*" Other distinctions were that subjects observed greater variation in PTM names ("*The versioning system for models seems less well-developed*"), and that subjects saw different kinds of names between PTMs and traditional packages: "*PyPI can be sensible names, based on archetypal action, or fun names, based on authors' aesthetics*".

Table 5. Induced themes for *how* PTM names differ from traditional names. Based on code-able responses from 34 participants.

Theme	# Participants (%)
Contain more semantic knowledge	30 / 34 (88%)
More variation in PTM naming practices	5 / 34 (15%)
Kinds: Archetypal vs. aesthetic names	4 / 34 (12%)

Table 6 summarizes the codes and response frequency related to *why* names differ between PTMs and traditional packages. The two main explanations were about evolution practices and

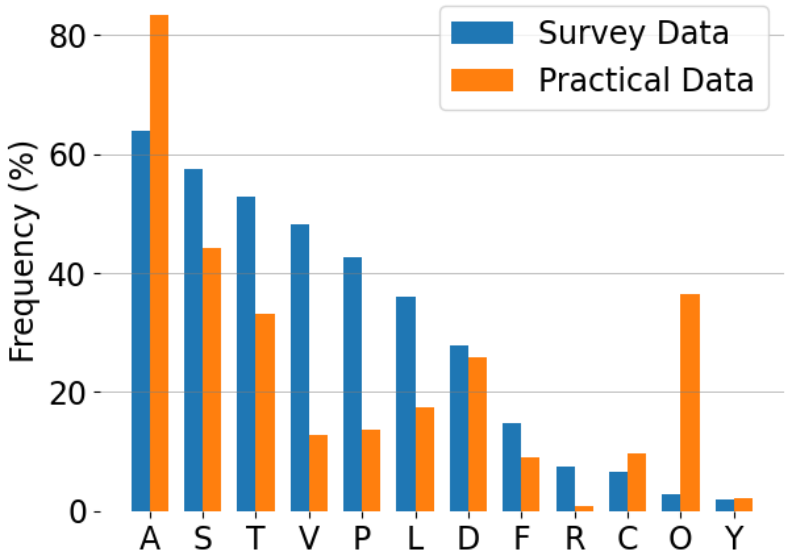


Fig. 5. The surveyed (blue) and measured (orange) distribution of naming elements. The x-axis refers to the codes from Table 1.

reuse practices. One subject described the influence of evolutionary practices on names as follows: "*[Traditional] packages are not usually as directly built on previous ones, with small extensions added like for PTM. Finetuning models generally means that the outputs are similar to the base, unlike with traditional packages.*" With respect to the information needed for reuse, a representative quote was: "*Traditional software packages are supposed to contain an entire library...[with] various functions...and configuration...PTMs are generally trained for a single objective...Therefore, [PTM] naming is critical for readability and avoiding mistakes.*". Due to space limitations, other quotes are in the supplement (**??**).

Table 6. Induced themes for *why* PTM names differ from traditional names. Based on code-able responses from 37 participants.

Theme	**# Participants (%)**
Evolution practice ("forking")	18 / 37 (49%)
Information needed for reuse	18 / 37 (49%)
Conventional location of semantic info.	6 / 37 (16%)
Definition of versioning	4 / 37 (11%)
No standardization, "Wild West"	5 / 37 (14%)
Other	7 / 37 (19%)

RQ2: What elements to include in PTM identifiers?

Finding 2: Participants prefer naming by architecture/function.

Survey participants preferred naming pre-trained models based on architectural characteristics and intended functions. There was less interest in incorporating training details.

Figure 5 illustrates survey participants' preferences for including specific elements in the naming of pre-trained models (PTMs). The elements most favored are related to the model's architectural lineage: "Architecture" (69, 63.9%), "Model size" (62, 57.4%), and "Task" (57, 52.8%). This highlights a strong preference for names that convey the model's high-level implementation details, facilitating easy identification and distinction, This is usually a starting point of PTM reuse as indicated by Jiang *et al.* [4]. A smaller yet significant number of users want low-level details, including model versioning (52, 48.1%), number of parameters (46, 42.6%), language (39, 36.1%), and training dataset (30, 27.8%). These elements are relevant to model selection and evaluation.

Comparing survey participant preferences with real PTM names (orange/blue bars in Figure 5), we see that naming practices do not always match naming preferences. Almost all models include Architecture (A) information, though only ~60% of respondents prefer it. Many respondents value information such as task (T), version (V), and parameter count (P); these elements are comparatively rare in real model names.

Finding 3: Design purpose is important in PTM naming.

The survey suggests a preference for names that combine details about both the implementation and the application or task (60.2%), while actual PTM packages tend to be named solely based on implementation units (59.0%).

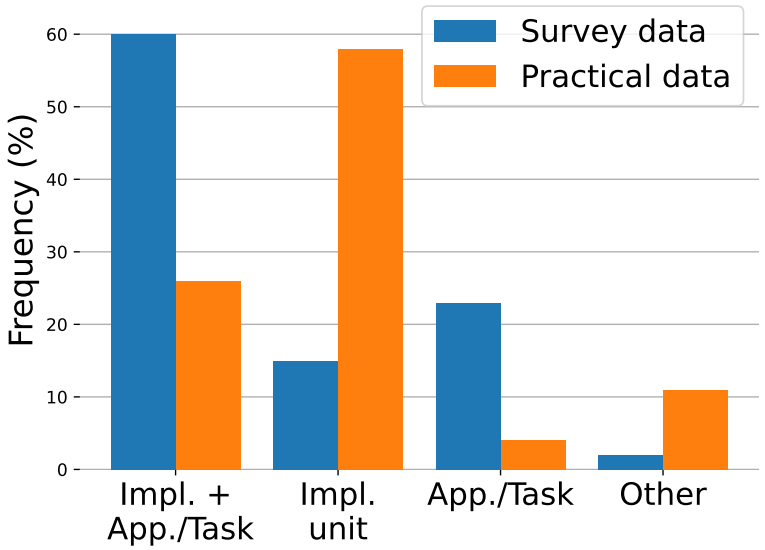


Fig. 6. The surveyed (blue) and measured (orange) distribution of naming conventions.

Figure 6 displays engineers' preferences for naming conventions in the context of reusing PTMs. The most preferred convention (58, 60.2%) combines "implementation + application/task" (*e.g.,* `Llama-2-7b-chat-hf`), with "Application/task" following (41, 23.3%). This suggests engineers favor names that reflect both the intended purpose and the implementation specifics of PTM packages. These patterns are rare in practice (only ~35% and ~5% of models) — PTM names tend to share only implementation details (59%). PTM users' preferences are misaligned with naming practices.

RQ3: How do engineers identify naming anomalies?

Finding 4: Users manually look for naming anomalies with model metadata and architecture inspection.

PTM users identify naming anomalies by reading both internal (69%) and external (11%) information provided by the PTM package. Around half of users (46%) inspect it, *e.g.,* via measurement or visualization.

A total of 52 respondents replied to the survey question corresponding to this RQ (§4.1.4). Table 7 summarizes the codes and response frequency for how to identify PTM naming anomalies. PTM users described two approaches: (1) reading internal/external information about the PTM; and (2) downloading and inspecting the PTM.

Table 7. Actions for identifying PTM naming anomalies, mentioned by multiple participants. The third column shows the number of participants who mentioned each process. 52 (out of 108) participants answered this question.

Action	# Participants (%)
Read metadata included with PTM	36 / 52 (69%)
Read external information about PTM	11 / 52 (21%)
Inspect the PTM	24 /52 (46%)

Most participants mentioned that they check for anomalies by reading information provided by the PTM packages. A large proportion of the participants (36/52, 69%) mentioned that they identify the naming anomalies by "*reading the model cards*" and other metadata included with PTM, and comparing the data with the elements in PTM names. The metadata available in the `config.json` file is another primary source for verifying the accuracy of naming elements. Those metadata are "*the ultimate source of truth because models can lack model cards*". Some participants (11/52, 21%) also mentioned that they refer to external information about PTM, including research papers, discussion forum, and issue reports. The external information can provide the users with more detailed information, *e.g.,* whether other people have already verified the naming elements for that PTM.

About half of the participants (24/52, 46%) mentioned that they inspect PTMs to identify naming anomalies. This inspection often involves "*visualizing/summarizing the model architecture*", "*running [the model] on actual data/examples*", and "*examining the memory usage*". They said that visualization or manual inspection of PTMs makes it "*obvious if the architecture is different to what the model name says*". When running the inference of the model, the users can verify if "*the memory usage matches their expectations*", or "*if there is any substantially decreased performance*". It is also obvious to find naming anomalies, particularly for the task and application goals, "*if the model is unable to perform the given task/achieve the mentioned goal*". Although costly, these hands-on approaches ensure that the model's capabilities match its description.

RQ4: Automatically identifying architectural anomalies?

Finding 5: Architectural information helps with naming anomaly detection.

We developed an automated tool to automatically identify naming anomalies. Our result shows that we can effectively (92.18% accuracy) detect naming anomalies based only on architectural information.

From RQ4, we learned that PTM users are concerned about naming anomalies, and that their current checking approaches are either cursory (reading metadata and external information)

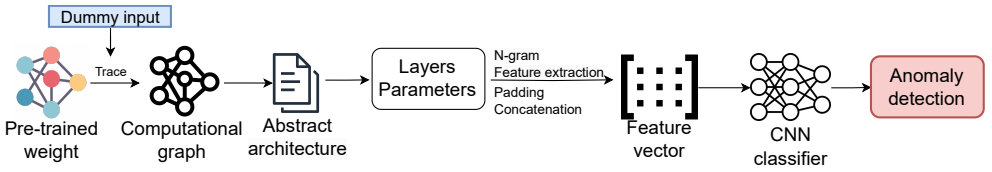


Fig. 7. To identify naming anomalies, we propose an algorithm of DNN ARchitecture Assessment pipeline (DARA). Our goal is to automatically identify naming anomalies. To achieve this goal, we automatically identify the architectural variations among multiple PTMs, subsequently train a classifier, and use that to identify naming anomalies. Anomalies are defined as mismatches between the prediction and the label assigned by the PTM creator. We evaluate DARA on two labels: `model_type` and `architecture`.

or costly (visual inspection or measurement of the PTM). Here we investigate the automatic detection of naming anomalies. Historical approaches to finding anomalous names in software have predominantly been rule-based [37–39]. Machine learning-based approaches are achieving state-of-the-art performance. For instance, Pradel *et al.* approached the problem of naming-based bug detection using binary classification [40]. They proposed a method for identifying and correcting naming-related issues in source code by mining name patterns from extensive codebases. This approach uses a classifier trained with a small dataset to achieve high precision in real-world repositories [41].

Inspired by these approaches, we propose a machine learning-based approach to detect anomalous names in PTM packages. We discuss feature selection (§5.1), feature extraction (§5.2), and evaluation on the PeaTMOSS dataset (§5.3).

5.1 Feature Selection

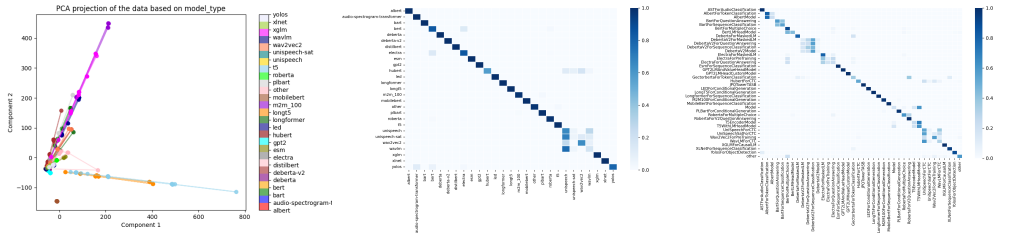
Table 8 presents a compilation of factors influencing the creation of new model types or architectures by PTM developers. More than one-third of the survey respondents believe that alterations such as modified training protocols, changes in input/output dimensions, layer additions or removals, modifications to the tokenizer, and shifts in the training dataset significantly contribute to the generation of new PTM names. If the typical engineer acts on these beliefs, then their naming function could be mathematically represented as:

$$f(\text{layer connections, layer parameters, tokenizer,}\\ \text{training regime, dataset}) = \text{model\_type/architecture} \tag{1}$$

Table 8. What changes necessitate new PTM names?

Theme	# Part. (%)
Modified training regime	43/108 (40%)
Modified tensor shape	34/108 (31%)
Modified layers in the main body	33/108 (31%)
Addition/deletion of layers in the main body	33/108 (31%)
Changed training dataset	31/108 (29%)
Modified tokenizer	31/108 (29%)
Addition/deletion of layers in IN/OUT layers	26/108 (24%)
Other	4/108 (4%)

The survey data suggests that the `model_type` and `architecture` are determined by several factors. However, prior study on PTM reuse suggested that there is a lot of missing metadata from the model cards on Hugging Face [9, 22]. Hugging Face does not provide enough details about



(a) **(b)** **(c)**

Fig. 8. (a) Visualization of selected PTM features, illustrating their sparse and high-dimensional distribution. Confusion matrix of evaluation results categorized by (b) model_type, and (c) architecture. The trend is visible from the colormap — see artifact for larger versions.

training regime and a lot of dataset information is missing, and the non-transparency reduced the trustworthiness and reusability of PTM packages [4]. The primary information provided by Hugging Face encompasses tokenizer details, pre-trained weights, and the labels for `model_types` and `architecture`, which leads to the following hypothesis:

Hypothesis: Using only architectural information, we can effectively detect naming anomalies.

To test this hypothesis, we evaluate an automated technique that extracts layer connection and parameter information, and then learns the relation between these features and the human labels from Hugging Face. We consider performance of over 75% (heuristic) as supportive of the hypothesis, indicating that further refinement could make the approach feasible for a model registry to use to automatically detect naming anomalies.

5.2 DNN Architecture Assessment (DARA)

5.2.1 Overview Figure 7 presents the DNN ARchitecture Assessment (DARA) workflow. The process begins with open-source pre-trained weights. First, we load each weight and feed dummy inputs to outline the graph architecture of each Pre-trained Model (PTM). Next, we transform the computational graph into an abstract architecture and implement n-gram feature extraction. Utilizing these features, we train a CNN classifier to identify naming anomalies.

5.2.2 Feature Extraction It is insufficient to identify naming anomalies solely based on the identifiers (§5.1). In this section, we present our feature extraction techniques, including graph conversion, delineation of abstract architecture, and n-gram feature selection.

Graph Conversion We developed an automated pipeline to transform pre-trained weights into computational graphs. Existing work shows that Pytorch is the dominant framework in Hugging Face [26], and using Hugging Face API we will load the PyTorch model by default [19]. Due to PyTorch's dynamic structure, we introduce *dummy inputs* to the models to construct and trace the computational graphs effectively [42]. We refined our automated pipeline by iteratively adjusting *dummy inputs*: if a conversion failed due to the input, we incorporated a new input type into our input list. Consequently, our finalized pipeline employs fixed language inputs while utilizing random tensors for models in the vision and audio domains. threat about this in §6

We conceptualized all neural network graphs as *directed acyclic graphs* (DAGs), enhancing the comprehensive and efficient application of algorithms. To handle cycles in models like LSTM [43], we remove upstream or backward edges, turning every network into a DAG with "complete

reachability." In this configuration, each node is connected through a path from an input to an output, guaranteeing that our algorithm covers all functional layers.

Abstract Architecture Prior work proposed *abstract neural network* (ANN) [44]. We proposed a similar concept, *abstract PTM architecture* (APTM). This new concept not only includes information about layers, parameters, and input/output dimensions but also incorporates detailed information about the connections between nodes, offering a fuller depiction of the model's architecture. This enhancement is pivotal for our n-gram feature selection method.

N-gram Feature Selection We propose a method for vectorizing PTM architectural information using two fundamental components that define their architecture: layer connections, and layer parameters. These aspects include the structural configuration of each PTM, such as pairs of connected layers (*e.g.,* `(Conv2D, BatchNormalization)`), and specific layer attributes (*e.g.,* `<kernel_size: [1, 1]>` in a Conv2D layer). Following prior works [45–47], we apply an N-gram feature extraction method to convert the architectural characteristics of each DNN into vector form. Specifically, we employ 2-grams for analyzing layer connections and 1-grams for detailing layer parameter information. Our goal is to find a good unsupervised feature for PTM architectures accurately and efficiently [48]. We also experimented with 3-gram methods for feature extraction; however, we found that the process was notably slow, and the resulting features were overly sparse and high-dimensional, suggesting this method is unsuitable for extracting features from PTM architectures. Subsequently, we apply padding to standardize the length of features across all PTMs. This innovative approach provides a structured and detailed basis for analyzing and comparing PTM architectures. The selected features of our sampled data is presented in Figure 8a.

5.2.3 CNN Classifier for Anomaly Detection We designed a Convolutional Neural Network (CNN) classifier for anomaly detection in the PTM dataset because this approach has been shown effective on similar tasks [49, 50]. We use an 80-20 split for training and evaluation. The classifier consists of four fully connected layers. We systematically implemented a grid search to optimize the hyperparameters for DARA. The final training parameters include 40 epochs, a learning rate of 1e-3, Cross Entropy loss, and a step-wise LR scheduler. Batch sizes are 256 for training and 32 for evaluation. We improved model validation using 5-fold cross-validation on the shuffled dataset.

5.2.4 Evaluation Methodology In this section, we demonstrate the *effectiveness* of our proposed anomaly detection method (DARA). We note that this is the first known approach to detect PTM naming anomalies so there is no baseline for comparison.

Our evaluation focuses on two key naming labels: `model_type` and `architecture` (§2.2). We evaluate DARA on a sampled data from the PeaTMOSS dataset by calculating the accuracy of the prediction of the trained classifier [9]. There are totally 132 architectures which has over 50 PTM instances. We randomly selected 50 architectures from the dataset (50/132, 38%). During feature extraction, we were unable to load the PTMs from four architectures, *e.g.,* `Bloom` [51], because their model sizes are too large to be loaded on a single `A100-80GB` GPU. This resulted in 46 unique architectures. We were also unable to load another 600 PTMs across these 46 categories because either our dummy inputs could not be used to those models, missing configuration files, or their architecture included customized code which we cannot fully trust. Overall, we collected 1,700 PTM packages from 46 architectures and 30 model types.

During training and evaluation of DARA, we changed the model_type and architectures of the categories which had less than 20 data as `Others`. This reduced the total number of architectures to 40, and model_types to 27.

5.3 Results and Analysis

Results: Our results indicated that DARA can proficienetly identify anomalies in `model_type` with an accuracy of 92.18% (Figure 8b). DARA's ability to discern anomalies in `architecture` is moderate, achieving an accuracy of 67.47% (Figure 8c).

Analysis: Figure 8b shows the cases where DARA cannot predict correctly in `model_type`. In fact, these models share almost the same architectural designs and the only differences are either different training regimes, different datasets, or using different representations. The n-gram feature extraction approach was not able to consider these distinct PTM features.

Figure 8c shows the cases where DARA cannot predict correctly in `architectures`. The same incorrect case arises for the four Audio models. Furthermore, we can also find that DARA can not correctly distinguish between different tasks as part of the `architecture`, such as `TokenClassification`, `QuestionAnswering`, and `SequenceClassification` in language models. Previous work indicates that the final layers of PTMs are highly task-specific [52]. Therefore, due to the high similarity of PTMs' features across these tasks, with differences confined to the final layers, DARA struggled to effectively differentiate between these categories.

The evaluation results of DARA indicates that DARA is capable enough by only using the architectural information (*e.g.,* layer connections and layer parameters) to detect naming anomalies, specifically in `model_type` and `architecture`. This result supports our hypothesis and we can *only* use the architectural information for naming anomaly detection.

Table 9. 5-fold cross-validation result for DARA.

Label Type	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Avg. Acc.
`Model_type`	92.65%	92.65%	93.53%	89.41%	92.65%	**92.18%**
`Architecture`	68.24%	67.65%	71.47%	60.88%	69.12%	**67.47%**

6 Threats to Validity

Construct We operationalize the concept of PTM names in terms of the identifier and architecture. Alternative operationalizations might incorporate package type or package contents. However, per the registry naming taxonomy (Figure 2), we believe our operationalization covers the elements that involve judgment on the part of the namer.

Internal A key internal threat to validity pertains to the reliability of measurements using GPT-based approach. Although DARA can identify naming anomalies, the features used by the DARA algorithm are weak. N-grams discard much information about the PTM architecture.

External In the practical measurements of naming practices, we only included attributes that we collected from PeaTMOSS, which is primarily focused on data from the largest PTM ecosystem, Hugging Face. Thus there is a slight risk of not being generalizable to other PTM ecosystems.

7 Discussion and Future Work

We discuss three directions for further work.

To Improve DARA Although DARA can identify real naming anomalies, the features used by the DARA algorithm are weak. For example, it cannot reflect the differences of training regime and datasets between different PTMs. In future work, we propose to develop stronger feature extraction methods by converting the architecture to embeddings. For example, contrastive learning [53] could be used to mitigate some of the weaknesses. Recent advancements have also unveiled techniques

for encoding directed graphs directly using transformers [54]. Applying these feature extraction techniques could improve DARA's ability to detect naming anomalies more effectively.

To Support Model Search and Reuse Our dataset of abstract PTM architectures includes meta-features suitable for various tasks, such as identifying potentially mislabeled architectures. Nguyen *et al.* demonstrated the utility of meta-features from PTM implementations on GitHub for enhancing AutoML performance [44]. Studies have also explored effective support and reuse of PTMs through novel model selection methods [55–57]. We propose adapting DARA to extract additional meta-features (*e.g.,* intermediate outputs) and devise techniques to support large-scale AutoML and efficient model selection, ultimately simplifying PTM adoption.

To Improve Trustworthiness Recent studies have highlighted growing concerns about vulnerabilities within PTM packages, such as "typosquatting" [58, 59] and "backdoors" [60, 61], exploiting these naming variances to introduce malware. Microsoft indicated that good naming practices can potential to expedite the detection and removal of malware [62]. Our study not only sheds light on the intricacies of PTM naming patterns but also sets the stage for future research to leverage additional meta-features of PTMs, thereby bolstering our ability to pinpoint and address PTM anomalies effectively.

8 Conclusion

Pre-Trained Models (PTMs) are a new unit of reuse in software engineering, where naming practices practices significantly impact their reusability. Similarly to prior reuse units, engineers struggle to name PTMs well. We conducted the first study of PTM naming practices by a mixed-method approach, including a survey and a repository mining study. Our finding shows that PTM naming is different from traditional naming in terms of evolution practices and reuse decisions. We also show that engineers prefer PTM naming with architecture/function, and design purpose is important in PTM naming. PTM users look for naming anomalies using metadata and manual inspection. Our tool applies a novel feature extraction method and indicated that we can effectively detect naming anomalies with only architectural information. There are many opportunities for future work to support engineers: in best practices for naming, in model search, and in improved automated naming anomaly detection.

Acknowledgments

This work was supported by gifts from Google and Cisco and by NSF awards #2107230, #2229703, #2107020, and #2104319.

References

[1] S. Farshidi, S. Jansen, and M. Deldar, "A decision model for programming language ecosystem selection: Seven industry case studies," *Information and software technology*, vol. 139, p. 106640, 2021.

[2] Y. Huang, F. Xu, H. Zhou, X. Chen, X. Zhou, and T. Wang, "Towards exploring the code reuse from stack overflow during software development," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, 2022, pp. 548–559.

[3] K.-K. Lau, "Software component models," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 1081–1082.

[4] 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.

[5] J. C. Davis, P. Jajal, W. Jiang, T. R. Schorlemmer, N. Synovic, and G. K. Thiruvathukal, "Reusing deep learning models: Challenges and directions in software engineering," in *Proceedings of the IEEE John Vincent Atanasoff Symposium on Modern Computing (JVA'23)*, 2023.

[6] Neptune.ai, "Hugging face pre-trained models: How to find the best one?" https://neptune.ai/blog/hugging-face-pre-trained-models-find-the-best, 2021.

[7] J. Tsay, A. Braz, M. Hirzel, A. Shinnar, and T. Mummert, "AIMMX: Artificial Intelligence Model Metadata Extractor," in *International Conference on Mining Software Repositories (MSR)*, 2020.

[8] S. Oladele, "Ml model registry: The ultimate guide," https://neptune.ai/blog/ml-model-registry, 2023, accessed: 2024-03-19.

[9] W. Jiang, J. Yasmin, J. Jones, N. Synovic, J. Kuo, Y. Tian, G. K. Thiruvathukal, , and J. C. Davis, "Peatmoss: A dataset and initial analysis of pre-trained models in open-source software," in *Proceedings of the 21th Annual Conference on Mining Software Repositories (MSR'24)*, 2024.

[10] G. Kotonya and J. Lee, "Teaching reuse-driven software engineering through innovative role playing," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 276–282.

[11] G. T. Heineman and W. T. Councill, "Component-based software engineering," *Putting the pieces together, addison-westley*, vol. 5, p. 1, 2001.

[12] R. C. Seacord, S. A. Hissam, and K. C. Wallnau, "Agora: A search engine for software components," *IEEE Internet computing*, vol. 2, no. 6, p. 62, 1998.

[13] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "Effective identifier names for comprehension and memory," *Innovations in Systems and Software Engineering*, vol. 3, no. 4, pp. 303–318, Nov. 2007. [Online]. Available: http://link.springer.com/10.1007/s11334-007-0031-2

[14] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the Influence of Identifier Names on Code Quality: An Empirical Study," in *2010 14th European Conference on Software Maintenance and Reengineering*, Mar. 2010, pp. 156–165.

[15] S. Henninger, "Using iterative refinement to find reusable software," *IEEE software*, vol. 11, no. 5, pp. 48–59, 1994.

[16] J. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," in *2017 IEEE 24th International conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2017, pp. 217–227.

[17] R. Alsuhaibani, C. Newman, M. Decker, M. Collard, and J. Maletic, "On the Naming of Methods: A Survey of Professional Developers," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021, pp. 587–599.

[18] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Mining java class naming conventions," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2011, pp. 93–102, iSSN: 1063-6773.

[19] H. Face, "Hugging face documentations," 2023. [Online]. Available: https://huggingface.co/docs

[20] 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.

[21] A. Bhatia, E. E. Eghan, M. Grichi, W. G. Cavanagh, Z. M. Jiang, and B. Adams, "Towards a change taxonomy for machine learning pipelines: Empirical study of ML pipelines and forks related to academic publications," *Empirical Software Engineering*, vol. 28, no. 3, p. 60, May 2023. [Online]. Available: https://link.springer.com/10.1007/s10664-022-10282-8

[22] W. Jiang, N. Synovic, P. Jajal, T. R. Schorlemmer, A. Tewari, B. Pareek, G. K. Thiruvathukal, and J. C. Davis, "Ptmtorrent: A dataset for mining open-source pre-trained model packages," *Proceedings of the 20th International Conference on Mining Software Repositories (MSR'23)*, 2023.

[23] 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.

[24] J. Castaño Fernandez, S. J. Martínez Fernández, J. Franch Gutiérrez, and J. Bogner, "Exploring the carbon footprint of hugging face's ml models: a repository mining study," in *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM): 26-27 Oct. 2023*. Institute of Electrical and Electronics Engineers (IEEE), 2023.

[25] 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.

[26] J. Castaño, S. Martínez-Fernández, X. Franch, and J. Bogner, "Analyzing the evolution and maintenance of ml models on hugging face," *arXiv preprint arXiv:2311.13380*, 2023.
[27] M. Taraghi, G. Dorcelus, A. Foundjem, F. Tambon, and F. Khomh, "Deep learning model reuse in the huggingface community: Challenges, benefit and trends," *arXiv preprint arXiv:2401.13177*, 2024.
[28] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, pp. 261–282, 2006.
[29] R. B. Johnson and A. J. Onwuegbuzie, "Mixed Methods Research: A Research Paradigm Whose Time Has Come," *Educational Researcher*, 2004.
[30] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 63–92.
[31] G. Garg, "The power of hugging face ai," https://medium.com/@gargg/the-power-of-hugging-face-ai-4f6558ee0874, 2023.
[32] "Sample size calculator," https://www.surveysystem.com/sscalc.htm, 2021.
[33] G. Guest, K. M. MacQueen, and E. E. Namey, *Applied thematic analysis*. sage publications, 2011.
[34] J. Saldaña, "The coding manual for qualitative researchers," 2021.
[35] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of Artificial General Intelligence: Early experiments with GPT-4," Apr. 2023. [Online]. Available: http://arxiv.org/abs/2303.12712
[36] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, "Why johnny can't prompt: how non-ai experts try (and fail) to design llm prompts," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–21.
[37] H. Liu, Q. Liu, C.-A. Staicu, M. Pradel, and Y. Luo, "Nomen est omen: Exploring and exploiting similarities between argument and parameter names," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1063–1073.
[38] M. Pradel and T. R. Gross, "Detecting anomalies in the order of equally-typed method arguments," in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, 2011, pp. 232–242.
[39] E. W. Høst and B. M. Østvold, "Debugging method names," in *European Conference on Object-Oriented Programming*. Springer, 2009, pp. 294–317.
[40] M. Pradel and K. Sen, "Deepbugs: A learning approach to name-based bug detection," *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 1–25, 2018.
[41] J. He, C.-C. Lee, V. Raychev, and M. Vechev, "Learning to find naming issues with big code and small supervision," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 296–311.
[42] Preferred Networks, "How computational graphs are constructed in pytorch," August 2021. [Online]. Available: https://pytorch.org/blog/computational-graphs-constructed-in-pytorch/
[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[44] G. Nguyen, M. J. Islam, R. Pan, and H. Rajan, "Manas: mining software repositories to assist AutoML," in *International Conference on Software Engineering (ICSE)*. Pittsburgh Pennsylvania: ACM, May 2022, pp. 1368–1380. [Online]. Available: https://dl.acm.org/doi/10.1145/3510003.3510052
[45] M. Ali, S. Shiaeles, G. Bendiab, and B. Ghita, "Malgra: Machine learning and n-gram malware feature extraction and detection system," *Electronics*, vol. 9, no. 11, p. 1777, 2020.
[46] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, "Bug or not? bug report classification using n-gram idf," in *2017 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2017, pp. 534–538.
[47] K. K. Sabor, A. Hamou-Lhadj, and A. Larsson, "Durfex: a feature extraction technique for efficient detection of duplicate bug reports," in *2017 IEEE international conference on software quality, reliability and security (QRS)*. IEEE, 2017, pp. 240–250.
[48] P. Zhu, W. Zuo, L. Zhang, Q. Hu, and S. C. Shiu, "Unsupervised feature selection by regularized self-representation," *Pattern Recognition*, vol. 48, no. 2, pp. 438–446, 2015.
[49] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 3009–3017.
[50] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *Proceedings of the 44th international conference on software engineering*, 2022, pp. 1356–1367.
[51] T. Le Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé *et al.*, "Bloom: A 176b-parameter open-access multilingual language model," 2022.
[52] A. Rogers, O. Kovaleva, and A. Rumshisky, "A primer in bertology: What we know about how bert works," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 842–866, 2021.

[53] R. Ni, M. Shu, H. Souri, M. Goldblum, and T. Goldstein, "The close relationship between contrastive learning and meta-learning," in *International Conference on Learning Representations (ICLR'21)*, 2021.
[54] S. Geisler, Y. Li, D. J. Mankowitz, A. T. Cemgil, S. Günnemann, and C. Paduraru, "Transformers meet directed graphs," in *International Conference on Machine Learning (ICML'23)*. PMLR, 2023, pp. 11 144–11 172.
[55] K. You, Y. Liu, J. Wang, and M. Long, "LogME: Practical Assessment of Pre-trained Models for Transfer Learning," in *International Conference on Machine Learning (ICML)*. PMLR, Jul. 2021, pp. 12 133–12 143. [Online]. Available: https://proceedings.mlr.press/v139/you21b.html
[56] K. You, Y. Liu, J. Wang, M. I. Jordan, and M. Long, "Ranking and Tuning Pre-trained Models: A New Paradigm of Exploiting Model Hubs," *The Journal of Machine Learning Research (JMLR)*, vol. 23, no. 1, pp. 9400–9446, Oct. 2021. [Online]. Available: http://arxiv.org/abs/2110.10545
[57] B. Lu, J. Yang, L. Y. Chen, and S. Ren, "Automating Deep Neural Network Model Selection for Edge Inference," in *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, Dec. 2019, pp. 184–193.
[58] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, C. Maurice, L. Bilge, G. Stringhini, and N. Neves, Eds. Cham: Springer International Publishing, 2020, pp. 23–43.
[59] M. Zimmermann, C.-A. Staicu, and M. Pradel, "Small World with High Risks: A Study of Security Threats in the npm Ecosystem," in *USENIX Security Symposium*, 2019.
[60] J. Boyens, A. Smith, N. Bartol, K. Winkler, A. Holbrook, and M. Fallon, "Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication (SP) 800-161 Rev. 1, May 2022. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-161/rev-1/final
[61] H. Langford, I. Shumailov, Y. Zhao, R. Mullins, and N. Papernot, "Architectural neural backdoors from first principles," *arXiv preprint arXiv:2402.06957*, 2024.
[62] Azure and contributors, "Abusing ml model file formats to create malware on ai systems: A proof of concept," https://github.com/Azure/counterfit/wiki/Abusing-ML-model-file-formats-to-create-malware-on-AI-systems:-A-proof-of-concept#detection, 2023.