Bridging the language gap: an empirical study of bindings for open source machine learning libraries across software package ecosystems

Hao Li · Cor-Paul Bezemer

Abstract Open source machine learning (ML) libraries enable developers to integrate advanced ML functionality into their own applications. However, popular ML libraries, such as TensorFlow, are not available natively in all programming languages and software package ecosystems. Hence, developers who wish to use an ML library which is not available in their programming language or ecosystem of choice, may need to resort to using a so-called binding library (or *binding*). Bindings provide support across programming languages and package ecosystems for reusing a host library. For example, the Keras .NET binding provides support for the Keras library in the NuGet (.NET) ecosystem even though the Keras library was written in Python. In this paper, we collect 2,436 cross-ecosystem bindings for 546 ML libraries across 13 software package ecosystems by using an approach called `BindFind`, which can automatically identify bindings and link them to their host libraries. Furthermore, we conduct an in-depth study of 133 cross-ecosystem bindings and their development for 40 popular open source ML libraries. Our findings reveal that the majority of ML library bindings are maintained by the community, with npm being the most popular ecosystem for these bindings. Our study also indicates that most bindings cover only a limited range of the host library's releases, often experience considerable delays in supporting new releases, and have widespread technical lag. Our findings highlight key factors to consider for developers integrating bindings for ML libraries and open avenues for researchers to further investigate bindings in software package ecosystems.

Hao Li and Cor-Paul Bezemer
Analytics of Software, GAmes And Repository Data (ASGAARD) Lab
University of Alberta, Edmonton, Alberta, Canada
E-mail: {li.hao, bezemer}@ualberta.ca

1 Introduction

Machine learning (ML) has become extremely popular in the last decade. Nowadays, there exist many ML applications in our daily lives, such as email spam filters, recommendation systems, and voice assistants. To provide ML features in an application, most developers rely on well-developed open source ML libraries, such as *TensorFlow* (Abadi et al., 2016) or *PyTorch* (Paszke et al., 2019). These open source ML libraries provide easy-to-use interfaces for software developers to use ML techniques in their projects. However, these libraries often target only one programming language and publish to one software package ecosystem. For example, *scikit-learn* (Pedregosa et al., 2011), a popular ML library which provides various ML algorithms, is written in Python and publishes to `PyPI`. Thus, Python developers can directly use the published *scikit-learn* package through `PyPI` but developers in other programming languages cannot use this library as easily.

Reusing existing libraries is a common practice in open source communities (Heinemann et al., 2011; Zaimi et al., 2015), as it can provide economic benefits by improving software quality and development productivity (Barros-Justo et al., 2018; Mohagheghi and Conradi, 2007). However, reusing libraries across programming languages is challenging due to programming language barriers. There exist several workarounds that allow a developer to use a library that was not written in their preferred language. First, they could choose an alternative but similar library that is written in their preferred language. However, such a similar library may not exist, and even if it does, it may only provide a subset of the required functionality. Another workaround is to develop the library from scratch, but this approach is error-prone and requires a large amount of work. Given the complexity of most ML libraries which require extensive expertise for development and maintenance, developing ML libraries from scratch in different programming languages is often impractical.

Bindings have emerged as a practical solution for reusing libraries across different programming languages. These bindings can act as wrapper libraries that convert interface calls, handle data type translations, and manage resource allocations. They typically rely on foreign function interfaces (FFIs) or specialized tools to interact with the original library, referred to as the *host library*. For example, the Rust programming language provides an FFI[1] to interact with C libraries, while tools like SWIG (Simplified Wrapper and Interface Generator)[2] support calling C/C++ libraries from various languages such as Python and JavaScript. Through bindings, developers can reuse the functionality of the host libraries without having to develop them from scratch.

Given the unique challenges and importance of ML libraries, it is crucial to investigate the prevalence and distribution of bindings specifically in this context. The *prevalence of bindings* refers to the number of available bindings for ML libraries in software package ecosystems, while the *distribution of*

---

[1] `https://doc.rust-lang.org/rust-by-example/std_misc/ffi.html`

[2] `https://www.swig.org`

bindings refers to how these bindings are spread across these ecosystems. To enable large-scale analysis of bindings and their host libraries, we introduce an approach called `BindFind` for automatically identifying bindings and extracting the names of the host libraries, which we refer to as *host names*. Using `BindFind`, we conduct a comprehensive investigation into open source ML libraries and their bindings across the ecosystems of 13 programming languages. Specifically, we seek to answer the following research questions (RQs):

RQ1. What is the prevalence of bindings for ML libraries in software package ecosystems, and how effective is `BindFind` in identifying them?

Understanding the prevalence of bindings informs developers about the option of reusing ML libraries in their preferred programming languages, while also providing researchers with valuable data for further study. We found that bindings and their host names can be very accurately identified in software package ecosystems using `BindFind`. We identified 2,436 bindings for 546 ML libraries across 13 software package ecosystems.

RQ2. How are ML libraries and their bindings distributed across ecosystems?

Analyzing the distribution of bindings reveals popular ecosystems and common pairs of ecosystems that provide bindings for the same host library. This information helps developers make informed choices about ecosystem support and encourages cross-ecosystem integration. The most frequent pair of ecosystems that provide bindings for the same ML library is `npm` with `PyPI`. While `PyPI` is the most popular ecosystem for ML libraries, `npm` is the most popular ecosystem for the bindings of ML libraries.

RQ3. How are cross-ecosystem bindings for popular ML libraries maintained?

Evaluating the maintenance quality of bindings highlights the risks and challenges, guiding better maintenance practices and decisions. Cross-ecosystem bindings often support a limited number of the host library's releases and suffer from high delays in supporting new releases, along with considerable technical lag. The situation is worse for bindings that are not maintained by the official library organizations.

The main contributions of this paper are as follows:

1. The first paper to study bindings for ML libraries within software package ecosystems.
2. A replication package[3] containing our dataset of 250,668 bindings (together with their host names) identified by `BindFind`. In addition, the replication package includes details on 546 ML libraries and their 2,436 bindings, as well as the results of our analysis in which we matched 3,277 versions of 133 bindings for 40 popular ML libraries (including 3,785 tags).

---

[3] `https://doi.org/10.5281/zenodo.12746638`

3. A framework for understanding how well the bindings of ML libraries are maintained. Our findings offer a foundation for developers to make informed decisions when selecting bindings.

Paper Organization. The rest of this paper is organized as follows. Section 2 gives background information about our study. Section 3 discusses related work. Section 4 presents our methodology. Section 5 presents the findings of our three RQs. Section 6 discusses the implications of our findings. Section 7 outlines threats to the validity of our study. Section 8 concludes the paper.

2 Background

In this section, we give background information about software package ecosystems and cross-ecosystem bindings for ML libraries.

2.1 Software Package Ecosystems

Traditionally, developers of open source libraries published their source code in a source code repository like Git (Chacon and Straub, 2014). Developers who wish to use those libraries could then download them directly from the source code repositories. However, developers had to resolve the library's dependencies and build the library manually. To help developers integrate a library more easily, the releases of software libraries can be published to a software package ecosystem. Open source libraries generally select a software package ecosystem to publish their *main package*, making it the official distribution channel for releases. Alternatively, open source libraries might continue to release versions directly through their Git repositories.

Most modern programming languages come with an official package manager and a package registry. This package manager, the package registry and all the packages are the key components of a software package ecosystem. Usually, a package manager helps developers to manage the dependencies of their applications, for example, by downloading a specific version of a dependency when the application is installed. In addition, package managers help developers publish their applications to the software package ecosystem. Most software package ecosystems of programming languages will provide a website for developers to search and browse the information of stored packages. Some examples of software package ecosystems are `Maven` for Java, `PyPI` for Python, and `npm` for JavaScript.

2.2 Cross-Ecosystem Bindings for ML Libraries

According to ISO/IEC TR 10182 (International Organization for Standardization, 2016), a language binding is defined as a specification that maps the

functional interface of a system facility (e.g., a software library) to a programming language. In practice, bindings serve as bridges between different programming languages, enabling software written in one language to use libraries developed in another. For example, *Tkinter*[4] provides a Python interface to the well-known *Tk* GUI toolkit, which is implemented in C. In the context of ML, libraries are often tailored to specific languages and ecosystems (Ben Braiek et al., 2018), such as *TensorFlow* and *PyTorch* mainly target Python and are distributed via `PyPI`. Through bindings, developers can use these state-of-the-art ML libraries across languages and ecosystems. This can be achieved through various mechanisms, including wrapper libraries that act as intermediaries, software development kits (SDKs) that bundle libraries with tools for development, or application programming interfaces (APIs) that facilitate interaction between different software components.

In our study, we use the term *cross-ecosystem bindings* to refer to packages that allow the host library to be utilized in other software package ecosystems. For example, *tensorflow* in `PyPI` and *tfjs* in `npm` (Smilkov et al., 2019) are cross-ecosystem bindings for the same ML library (*TensorFlow*), even though they have different names and reside in different software package ecosystems. Since *tensorflow* in `PyPI` and *tfjs* in `npm` are both maintained by the official organization of the *TensorFlow* host library, we consider these two bindings as *officially-maintained bindings*. Officially-maintained bindings are those managed by the same organization that develops the host library, based on the management and ownership of the binding's source code repository rather than the funding source. In contrast, *community-maintained bindings* are managed by individuals or organizations that are not directly associated with the host library's official organization. For example, *TensorFlow.NET* in `NuGet` is a community-maintained binding since its owner is different from that of *TensorFlow*.

Host libraries and their cross-ecosystem bindings do not necessarily follow the same release schedules and/or strategies. For instance, a cross-ecosystem binding may choose to only support a portion of the releases of its host library. Also, after the host library publishes a release, there may be a delay before a cross-ecosystem binding supports that release (if at all). In addition, cross-ecosystem bindings might lag in version compared to the latest release of the host library.

3 Related work

In this section, we discuss prior empirical studies of ML libraries, and related work on software ecosystems and foreign function interfaces.

---

[4] `https://docs.python.org/3/library/tkinter`

3.1 Empirical Studies of ML Libraries

The increasing popularity of ML has led to significant research interest in
understanding the development, usage patterns, and challenges of ML libraries.
However, there is a lack of research on the usage of ML bindings across different
programming languages. This is the first paper to focus on bindings for ML
libraries, which play a crucial role in bridging different programming languages
and enabling the wider adoption of ML libraries.

Dilhara et al. (2021) conducted a large-scale empirical study, revealing a
significant rise in ML library adoption and identifying common usage patterns
and challenges. Further analysis by Gao et al. (2024) focused on the supply
chain structure and evolution of the *TensorFlow* and *PyTorch* packages within
`PyPI`, uncovering domain specialization, community clusters, and the reasons
for packages leaving the supply chains.

Several studies focused on the problems that developers could face when
using ML libraries. Islam et al. (2019) mined Q&A of ten ML libraries on
`StackOverflow`, and reported that three types of problems occurred frequently
(i.e., type mismatch, data cleaning, and parameter selection). Wei et al. (2022)
introduced FIMAX to improve API recommendations for Python-based ML
libraries based on extracted questions from Stack Overflow related to six pop-
ular ML libraries. Lei et al. (2023) identified seven primary reasons why ML
projects built on *TensorFlow* and *PyTorch* often encounter compatibility is-
sues due to library version changes, causing code to fail in some projects even
when using the same library API.

Several researchers have conducted comparison studies of multiple ML li-
braries. Grichi et al. (2020) compared ten multi-language ML frameworks with
ten multi-language traditional systems and reported that maintainers of these
ML frameworks need more time to accept or reject a pull request than tradi-
tional systems. Guo et al. (2019) compared the development and deployment
processes of four ML libraries under the same configuration for training of the
same models. They found that using different ML libraries can lead to different
levels of accuracy. Han et al. (2020) collected projects that depend on *PyTorch*,
TensorFlow, and *Theano* on GitHub, and observed four frequent applications
(i.e., image and video, NLP, model theory, and acceleration). In addition, most
projects depend on these three libraries directly instead of transitively.

3.2 Software Ecosystems

"Software ecosystems" are studied from several angles and even using different
definitions (Franco-Bedoya et al., 2017; Manikas, 2016; Mens et al., 2014). The
"software package ecosystems" term covers a subset of the software ecosystems
term. In this paper, we study ML libraries that can be found across multiple
software package ecosystems. These package ecosystems have formal processes
for maintenance and official distribution channels for releases, aligning with the
focus of our study. Our paper is the first to focus on cross-ecosystem bindings.

Prior research has identified cross-ecosystem packages by finding those sharing a common source code repository (Constantinou et al., 2018; Kannee et al., 2023). These studies examined packages that are maintained within a single repository but are distributed across multiple ecosystems. However, they did not directly address the concept of bindings. While these studies might have unintentionally included some officially-maintained bindings within the same repository, they overlooked both community-maintained bindings and officially-maintained bindings hosted in different repositories. Our findings in Section 5.1 demonstrate that 94% of the ML libraries bindings are community-maintained, underscoring a key difference in our work.

Many studies have focused on software package ecosystems. In our prior work (Li et al., 2023), we studied the release-level deprecation mechanism in `Cargo` (Rust) ecosystem and found that the deprecated releases propagate through the dependency network and lead to broken releases. German et al. (2013) mined packages in `CRAN` and reported that most dependencies point to a core set of packages in the ecosystem. This phenomenon was also observed in another active software package ecosystem – `npm` (Wittern et al., 2016). Cogo et al. (2021) studied dependency downgrades in `npm` and found three reasons behind the downgrades: defects, unexpected changes, and incompatibilities. Constantinou and Mens (2017a,b) studied social aspects in ecosystems and found that the developers are more likely to abandon an ecosystem if they do not participate in the community, and the Ruby ecosystem is being abandoned. Kula et al. (2018) proposed a model for visualizing dependencies in ecosystems, and show that `CRAN` packages tend to use the latest releases, but `Maven` packages stay with the older versions. Decan and Mens (2020) investigated package releases in three software package ecosystems and observed that most pre-releases do not become $\geq 1.0.0$ releases. Moreover, software package ecosystems have different practices, policies, and tools for handling breaking changes (Bogart et al., 2016, 2021).

In addition, researchers studied other types of software ecosystems. Blincoe et al. (2015, 2019) proposed a reference coupling method to identify software ecosystems in GitHub as well as the dependencies in the ecosystems. Osman and Baysal (2021) identified the Bitcoin software ecosystem in GitHub and assessed it as a healthy ecosystem. Furthermore, many researchers studied the health of software ecosystems (Jansen, 2014; da Silva Amorim et al., 2016). Kula and Robles (2019) investigated four abandoned software ecosystems and observed that all these ecosystems had a successor or their components were adopted by other systems. Bavota et al. (2013) found that projects in the Apache ecosystem get updates when the dependencies published releases for breaking changes or bug fixes. de Souza et al. (2016) studied social aspects in proprietary mobile software ecosystems and observed that most developers chose a specific ecosystem based on others' recommendation. Researchers also studied information security and business factors in mobile software ecosystems (Steglich et al., 2020a,b).
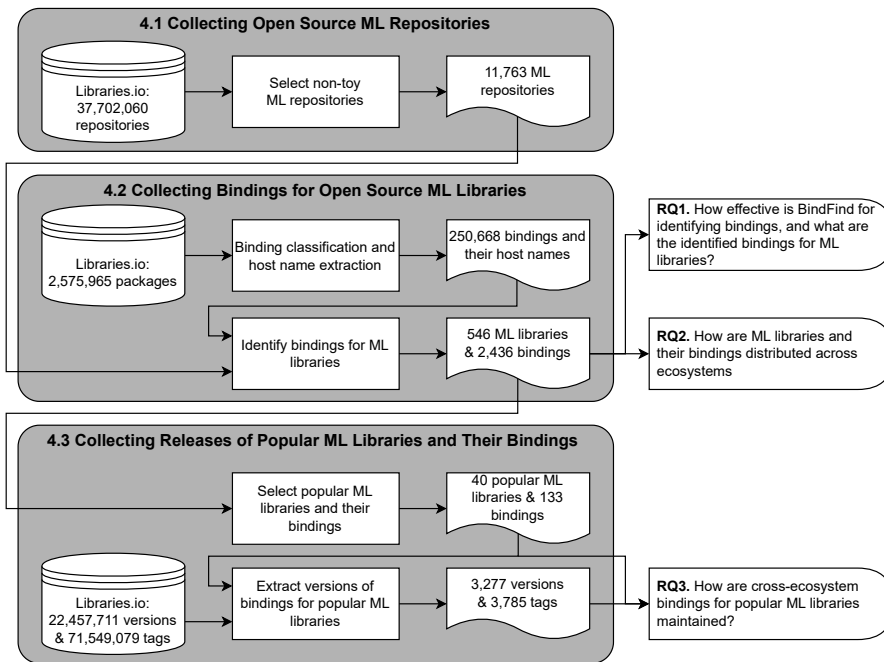
Fig. 1: Overview of our methodology.

3.3 Foreign Function Interfaces

Foreign function interfaces (FFI) bridge the gap between different programming languages and allow developers to reuse libraries written in other languages. To verify the correctness of existing bindings, Furr and Foster (2008) presented a static checking system that analyzes both bindings and their host library. In addition, Lee et al. (2010) built bug detection tools for the FFI in Jave and Python by performing dynamic analysis. Nakata et al. (2011) categorized link models and fault models of FFI and proposed a logging framework to track the information flow for each model.

Moreover, wrapping up a function to call a library from another programming language is not always applicable, Chiba (2019) proposed a framework based on code migration to solve this problem. Besides writing the codes of FFI manually, Finne et al. (1998) used an interface definition language to allow Haskell to communicate with both C and COM. In addition, Reppy and Song (2006) developed a tool to generate foreign interfaces for high-level languages to use the libraries written in C.

Table 1: Overview of the `Libraries.io` dataset

(a) Repositories and tags in Git hosts

Platform	# Repos	# Tags
GitHub	36,567,566	58,296,891
GitLab	864,563	12,248,518
Bitbucket	269,931	1,003,670
Total	37,702,060	71,549,079

(b) Packages and versions in software package ecosystems

Ecosystem	Language	# Packs	# Versions
npm	JavaScript	1,277,221	11,400,714
Packagist	PHP	313,575	1,766,576
PyPI	Python	232,050	1,752,770
NuGet	C#	199,671	2,445,003
Maven	Java	184,890	2,799,513
RubyGems	Ruby	161,650	1,055,874
CocoaPods	Objective-C	68,085	365,782
CPAN	Perl	37,496	290,847
Cargo	Rust	35,695	195,011
Clojars	Clojure	24,295	116,945
CRAN	R	16,710	94,716
Hackage	Haskell	14,484	98,572
Pub	Dart	10,143	75,388
Total		2,575,965	22,457,711

4 Methodology

In this section, we introduce the methodology of our study on popular ML libraries and their cross-ecosystem bindings. Figure 1 gives an overview of our methodology.

4.1 Collecting Open Source ML Repositories

We used the `Libraries.io` dataset (Katz, 2020) which was updated on January 12, 2020 as our primary data source. This dataset contains information (e.g., tags, owners, keywords) on 37,702,060 repositories from three prominent Git hosting services. These services host the actual code of open source ML libraries and facilitate version control, developer collaboration, and other functions. Since these hosts are widely used, they may also contain personal projects, documentation, and experimental code. Table 1 outlines the distribution of repositories and tags among these Git hosts.

Following the keyword-matching approach proposed by Ben Braiek et al. (2018) to extract ML projects from GitHub, we employed a similar approach to identify ML repositories. This approach involved scanning the "Description" and "Keywords" fields of repositories in the dataset for relevant keywords. The keywords included:

"machine learning", "deep learning", "statistical learning", "neural network", "supervised learning", "unsupervised learning", "reinforcement learning", and "artificial intelligence."

We crafted regular expressions to accommodate variations in keyword formatting, including the presence of underscores, hyphens, and commas (e.g., "machine_learning"). To ensure we selected representative repositories, we filtered out those with fewer than 5 stars, marked as inactive in the "Status" field, or indicated as forks. We chose the same cut-off of 5 stars as previous studies (Bernardo et al., 2024; Kochanthara et al., 2022; OBrien et al., 2022; Song et al., 2024) which have shown this threshold to be effective. Ultimately, we extracted 11,763 ML repositories from the dataset.

4.2 Collecting Bindings for Open Source ML Libraries

The `Libraries.io` dataset includes information (e.g., released versions, creation dates, dependencies) on 4,612,919 packages from 38 software package ecosystems. As described in Section 2, ML libraries typically have their source code managed on Git hosts and publish packages to these ecosystems. Our analysis focused on 13 selected ecosystems, excluding ecosystems that: (1) focus on a specific domain, such as `Sublime` and `WordPress`, (2) those with a very small number of packages, such as `Shards` (33 packages) and `PureScript` (384 packages), and (3) those that do not store information about releases, such as `Go`. Also, we excluded ecosystems that contain duplicated packages of other ecosystems, for example, most packages in `Bower` can be found in `npm`. Table 1 shows the supported programming language, the number of packages, and the number of releases in these 13 ecosystems.

4.2.1 Binding classification and host name extraction

To automatically identify bindings in package ecosystems and extract their corresponding host names, we propose `BindFind`. `BindFind` employs natural language processing (NLP) techniques, specifically leveraging BERT (Bidirectional Encoder Representations from Transformers) models (Devlin et al., 2019). As shown in Figure 2, we conceptualize the problem as an extractive question-answering (QA) task, akin to the methodology used in the Stanford Question Answering Dataset (SQuAD) v2.0 (Rajpurkar et al., 2018). In this framework, the description of a package serves as the context for querying the model about the package's host name. This description specifically refers to the textual summary or overview provided by the package ecosystem in the Libraries.io dataset (Katz, 2020). For instance, it can be the "description"

fields in `Maven` and `RubyGems`.[5] In other ecosystems, this information might be found in the "readme" section of `NuGet`[6] or the short description of `PyPI`.[7]

The BERT-like model determines the precise locations (i.e., the start and end positions) of the host name within the given context. Notably, all the inputs are tokenized before being fed into the model, and the start and end positions refer to the tokens instead of the original input. If the start and end positions point to the [CLS] token, or if the positions are invalid (e.g., the start position is after the end position), we conclude that the model did not identify the repository as a binding. This approach adeptly handles both scenarios where questions are answerable and unanswerable, reflecting real-world scenarios where some packages might not be bindings and thus not have a host name to extract.

Studied models. We selected several variations of BERT models, including the original BERT (Devlin et al., 2019), DistilBERT (Sanh et al., 2019), AL-BERT (Lan et al., 2020), and RoBERTa (Liu et al., 2019). These models were chosen for their proven efficacy in QA benchmarks (Rajpurkar et al., 2016, 2018).

Data preparation. We manually labeled 2,546 packages to determine whether they are bindings and, if so, to identify their host names. The dataset contained 2,054 non-bindings (80.7%) and 492 bindings (19.3%). To ensure a fair evaluation, we balanced the validation and test sets by randomly selecting an equal number of bindings and non-bindings (50% each). This approach ensures that performance metrics are not skewed by the imbalanced nature of the dataset. Both the validation and test sets were balanced, each containing 100 samples with an equal distribution of 50 bindings and 50 non-bindings. The remaining 2,346 samples (392 bindings and 1,954 non-bindings) formed the training set. During the training process, we trained the studied models on the training set and used the validation set for hyperparameter tuning and model selection. The test set was reserved for the final evaluation of model performance.

Evaluation metrics. To assess the effectiveness of `BindFind`, we used the **F1 score**, **precision**, and **recall** (Godbole and Sarawagi, 2004) to evaluate the performance of classifying whether a package is a binding. For the extraction of host names within identified bindings, we applied **exact match** (EM) accuracy and (macro-average) **F1 score** specific to QA tasks (Rajpurkar et al., 2018). EM is a strict metric where any deviation from the exact answer results in a score of 0 for that sample. The F1 score evaluates performance by considering predictions and ground truths as collections of tokens. The reported results are averaged over all of the samples.

---

[5] Examples: `https://search.maven.org/artifact/de.vorb/jtesseract/0.0.4/jar` and `https://rubygems.org/gems/ruby-opencv/versions/0.0.18`

[6] `https://www.nuget.org/packages/Keras.NET`

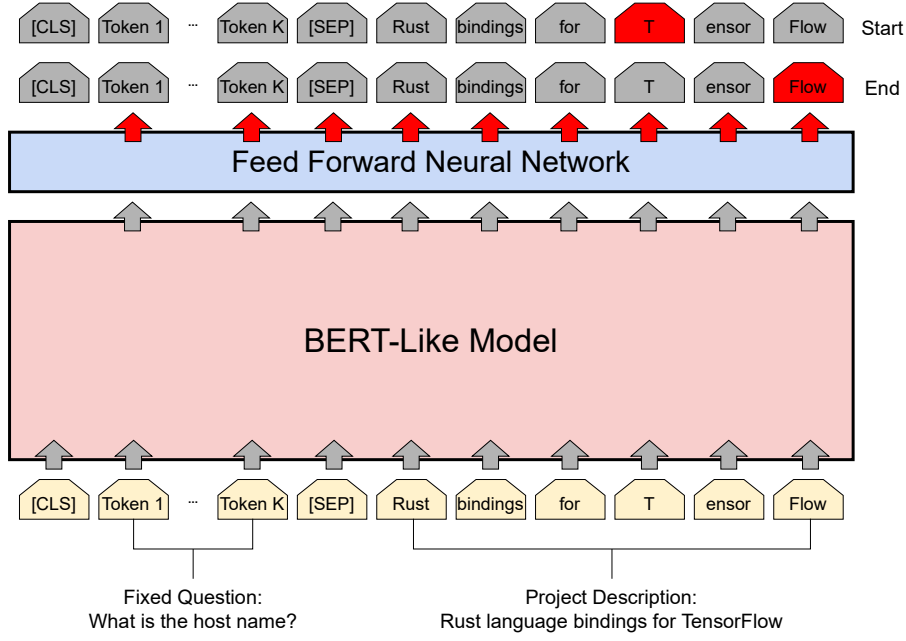[7] `https://pypi.org/project/opencv-python`

Fig. 2: The model structure of `BindFind` for binding classification and host name extraction, illustrated using an example.

$$EM = \frac{\text{Number of exact matches}}{\text{Number of examples that are bindings}} \tag{1}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2}$$

where

$$Precision = \frac{\text{Number of accurately identified tokens}}{\text{Number of tokens in the prediction}} \tag{3}$$

$$Recall = \frac{\text{Number of accurately identified tokens}}{\text{Number of tokens in the ground truth}} \tag{4}$$

4.2.2 Identify bindings for ML libraries

We processed 2,575,965 packages from the 13 studied software package ecosystems and identified a total of 250,668 bindings with their corresponding host names. To identify bindings for the 11,763 ML repositories, we deployed a string-matching algorithm that compared the names of ML repositories with the extracted host names, resulting in 3,360 matches between bindings and

983 ML repositories. We manually reviewed these matches, removing repositories containing only tutorials or experimental code, and filtering out duplicate repositories of the same ML library. After refinement, we identified a total of 2,436 bindings for 546 ML libraries.

It is important to note that some of the 546 ML libraries might publish official packages into ecosystems. These published packages can be either normal packages or bindings (already identified by `BindFind`). For example, *PyTorch* publishes its main package *torch* in `PyPI`,[8] which is not considered a binding as stated in their documentation (PyTorch, 2022). To study the distribution of ML libraries and their bindings across ecosystems in RQ2, we included officially published packages alongside identified bindings. We used two criteria: (1) they specified the ML library's Git repository as their source code repository, and (2) they shared the same homepage URL as the ML library. Using these criteria, we identified 775 packages officially published packages by 202 out of the 546 ML libraries.

4.3 Collecting Releases of Popular ML Libraries and Their Bindings

To focus our analysis in RQ3 on widely adopted libraries, we filtered the 546 ML libraries with 2,436 bindings based on the number of stars and selected 127 ML libraries with more than 1,000 stars. Though we acknowledge that stars do not provide a complete picture of real-world usage, they are commonly seen as a proxy for the popularity of a project within the software engineering domain (Borges et al., 2016; Fang et al., 2022; Han et al., 2019; Wolter et al., 2023; Xia et al., 2023). For instance, *TensorFlow*'s binding *tfjs* has gained over 17,000 stars on GitHub,[9] suggesting significant attention from developers. For further analysis, we manually reviewed these libraries and their bindings to perform several refinements. We excluded supporting packages for the actual bindings within the same ecosystem, along with bindings that are either work-in-progress or have only placeholder/invalid releases, e.g., *OpenCV*'s binding in `Pub`.[10] Following these steps, we obtained a final set of 40 popular ML libraries (as shown in Table 2) with 133 bindings.

5 Results

This section presents the results of our three RQs. For each RQ, we present the motivation, approach, and findings.

---

[8] `https://pypi.org/project/torch`

[9] `https://github.com/tensorflow/tfjs`

[10] `https://pub.dev/packages/flutter_opencv_plugin`

Table 2: Basic information about the popular ML libraries that have cross-ecosystem bindings.

ML library	# Eco	# Stars	Description
Alluxio	2	4,449	Data orchestration for ML in the cloud
BerryNet	1	1,150	Deep learning gateway on Raspberry Pi
bert-as-service	2	6,379	Sentence vector mapping with BERT
BigDL	2	3,177	Distributed deep learning library for Apache Spark
Bullet	2	5,798	Physics simulation for RL
Caffe	3	29,655	Deep learning library
CatBoost	2	3,760	Gradient Boosting on Decision Trees
Deeplearning4j	2	11,328	Deep learning library for Java
DeepSpeech	4	12,710	A speech-to-text engine based on TensorFlow
dlib	5	8,412	A toolkit for real-world ML applications
DyNet	2	2,867	Dynamic Neural Network Toolkit
H2O	3	4,513	A platform for distributed ML
ImageAI	2	4,252	A library for deep learning and computer vision
Keras	3	45,995	A framework to provide human-friendly APIs based on TensorFlow
libpostal	6	2,171	An NLP library for address parsing and normalizing
LightGBM	3	10,242	A gradient boosting framework
MITIE	2	1,820	An NLP library for information extraction
MLflow	3	5,459	A ML lifecycle platform
mlpack	2	3,016	A library to provide ML algorithms
ncnn	1	7,930	Neural network inference
NLTK	3	8,498	An NLP library
NNPACK	1	1,085	Neural network acceleration
NNVM	1	1,586	Compiler for neural nets
ONNX Runtime	4	1,561	A runtime engine for ONNX models
OpenAI Gym	7	19,351	A toolkit for developing and comparing RL algorithms
OpenCV	10	41,126	A computer vision library
OpenFace	1	12,847	Face recognition with deep learning
OpenPose	1	15,532	Multi-person keypoint detection
Porcupine	3	1,853	A library for lightweight wake word detection
PredictionIO	8	12,226	A ML server for infrastructure management
PyTorch	5	35,004	A ML framework
Rasa	2	7,436	A ML framework for automating conversations based on text and voice
scikit-learn	3	38,756	A framework to provide ML algorithms
Seldon Core	2	1,296	An MLOps framework based on Kubernetes
spaCy	2	15,161	An NLP library
TensorFlow	11	139,939	A ML framework
Tesseract OCR	8	32,078	An OCR engine that uses deep learning
Vowpal Wabbit	8	6,767	Techniques to solve interactive ML problems
Weld	2	1,261	A library for data analy
XGBoost	6	17,996	A gradient boosting framework

5.1 RQ1: What is the prevalence of bindings for ML libraries in software package ecosystems, and how effective is `BindFind` in identifying them?

Motivation. Researchers have studied different aspects of open source ML libraries (Ben Braiek et al., 2018; Dilhara et al., 2021), but there is a gap in understanding the prevalence of bindings for these libraries. Bindings enable ML library reuse across programming languages, facilitating broader ML integra-

Table 3: Performance comparison of `BindFind` with different BERT models in bindings classification and host name extraction on the test set. (U: Uncased; C: Cased; Prec: Precision; Rec: Recall)

Family	Variant	Binding Classification			Host Name Extraction	
		F1	Prec	Rec	F1	Exact Match
DistilBERT	Base (U)	0.903	0.977	0.840	0.806	0.780
	Base (C)	0.936	1.000	0.880	0.802	0.740
BERT	Base (U)	0.925	1.000	0.860	0.770	0.760
	Large (U)	0.925	1.000	0.860	0.799	0.760
	Base (C)	0.947	1.000	0.900	0.803	0.760
	Large (C)	0.936	1.000	0.880	0.767	0.720
ALBERT	Base	0.918	0.938	0.900	0.824	0.760
	Large	0.925	1.000	0.860	0.818	0.800
	XLarge	0.876	1.000	0.780	0.696	0.640
	XXLarge	0.959	0.979	0.940	0.887	0.840
RoBERTa	**Base**	**0.970**	**0.980**	**0.960**	**0.889**	**0.860**
	Large	0.969	1.000	0.940	0.869	0.840

tion in diverse projects. However, the absence of comprehensive tools for identifying bindings has been a major obstacle in this area of research. `BindFind` fills this gap by offering a method for classifying bindings and extracting host names, thereby enabling future research and practical applications. The findings from this RQ will (1) inform developers about the availability of bindings for ML libraries, (2) open new research opportunities for researchers to study bindings in software ecosystems, and (3) provide a tool for repository hosting platforms (e.g., GitHub) to discover bindings for assisting developers.

Approach. We conducted a comparative analysis of various BERT models as described in Section 4.2. Subsequently, the most effective model was selected and utilized by `BindFind` to perform inference across all the packages in the 13 studied software package ecosystems. The distinction between officially-maintained and community-maintained bindings was established through an automatic examination of their association with the host ML libraries. Bindings sharing the same source code repository, organization name, or homepage URL as the host ML library were categorized as officially-maintained. Conversely, those lacking such affiliations were categorized as community-maintained.

Findings. **5% of the ML repositories have bindings in software package ecosystems and a vast majority (94%) of them are community-maintained bindings.** We found that 5% (546 out of 11,763) of the ML repositories are ML libraries with bindings in software package ecosystems. Among the 2,436 identified bindings for these ML libraries, a staggering 94% (2,292 out of 2,436) of the bindings are maintained by the community. Conversely, only 6% (144 out of 2,436) of the bindings are maintained by the official organization. Notably, 58% (84 out of 144) of these officially-maintained bind-

ings share the same source repository as their host ML libraries, while 42% (60 out of 144) are hosted under the same organizational umbrella but in separate repositories. For instance, *PyTorch* has officially published a binding (PyTorch, 2021) in `Cocoapods` for iOS, and *tfjs-node* serves as an official binding for *TensorFlow* within the `NPM` ecosystem (Smilkov et al., 2019), with its source repository is maintained by the *tensorflow* organization on GitHub. Additionally, we observed the transition of some community-maintained bindings to official organizations, such as the Python binding for *OpenCV* (OpenCV, 2021).

BindFind, powered by the RoBERTa base model, demonstrates exceptional performance in the tasks of binding classification and host name extraction, achieving F1 scores of 0.970 and 0.889 respectively. As illustrated in Table 3, the RoBERTa base model exhibits superior performance in both the binding classification and host name extraction tasks. Notably, all evaluated models achieved an F1 score above 0.870 in binding classification, indicating a generally high level of accuracy across different architectures. Furthermore, the results demonstrate that larger models do not always guarantee better performance. For instance, the BERT base (cased) model outperforms its larger variant, and the RoBERTa base model surpasses the RoBERTa large model in both tasks.

> **Takeaway of RQ1**
>
> Using `BindFind`, we found that 5% of the ML repositories have bindings in software package ecosystems. `BindFind` demonstrated high effectiveness in identifying bindings (with a 0.970 F1 score) and extracting their host names (with a 0.889 F1 score).

5.2 RQ2: How are ML libraries and their bindings distributed across ecosystems?

Motivation. Existing research shows that Python is the most popular language for ML libraries (Ben Braiek et al., 2018), but it remains unclear which other ecosystems provide good support for ML libraries through bindings. Also, we aim to explore whether certain ecosystems or combinations of ecosystems are favoured by these bindings in this RQ. By investigating the distribution of bindings across ecosystems, we can identify patterns of ecosystem support and cross-ecosystem interaction. The findings from this RQ will (1) provide developers with insights into ecosystems that provide good support for ML other than Python and (2) motivate researchers to further explore the adoption of bindings for ML libraries across ecosystems.

Approach. We analyzed 2,436 bindings for 546 ML libraries, with a subset of 202 out of these 546 libraries having officially published packages (not bindings) across ecosystems (as detailed in Section 4.2). Our analysis focused on their distribution across 13 software package ecosystems. To better understand
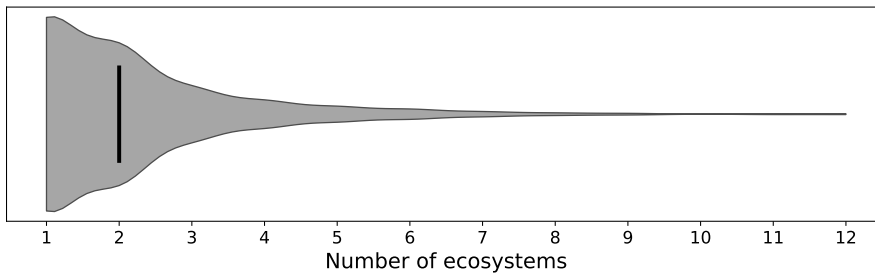
Fig. 3: The distribution of the number of software package ecosystems supported by ML libraries with bindings.

in which combinations of software package ecosystems these libraries reside, we counted the *ecosystem-pairs* for each library. An ecosystem-pair is counted for each pair of ecosystems supporting the same library. For example, if a library is supported in PyPI, npm, and NuGet, we count three ecosystem-pairs: PyPI-npm, PyPI-NuGet, and npm-NuGet. If an ecosystem-pair appears more frequently than others, it implies that those two ecosystems are more likely to be supported together by ML libraries. The frequency of these ecosystem-pairs reveals the prevalence of cross-ecosystem interactions, which is not apparent when considering individual ecosystems alone. Prevalence ecosystem-pairs demonstrate the flexibility developers have when switching between languages to use ML libraries in their projects.

Findings. **ML libraries with bindings typically span across a median of two software package ecosystems.** Our analysis revealed that ML libraries with bindings are typically supported across a median of two software package ecosystems. As illustrated in Figure 3, 55% (302 out of 546) of the studied ML libraries extend their reach by residing in multiple ecosystems. For example, the NLP library *spaCy* has bindings available in both PyPI and npm. Notably, the library with the broadest ecosystem presence is *OpenCV* which is available in 12 different ecosystems, followed by *TensorFlow* which is available in 11 ecosystems. Among the ML libraries with bindings that are found in a single ecosystem (244 out of 546 libraries), 71% (174 out of 244) consists of libraries with only community-maintained bindings, lacking official packages or officially-maintained bindings. The remaining cases (70 out of 244) comprise libraries that have officially-maintained bindings.

npm is the leading ecosystem for hosting bindings for ML libraries and the most common combination of bindings is npm with PyPI. Regarding bindings for ML libraries, we found that 53% (292 out of 546) of the libraries have bindings in npm. This is closely followed by PyPI in which bindings for 41% (225 out of 546) of the libraries are hosted. Figure 4 gives an overview of the ecosystem-pairs of ML libraries with bindings across multiple ecosystems (302 out of 546). The PyPI-npm pair is identified as the most prevalent combination, as it is supported by 139 out of 302 ML
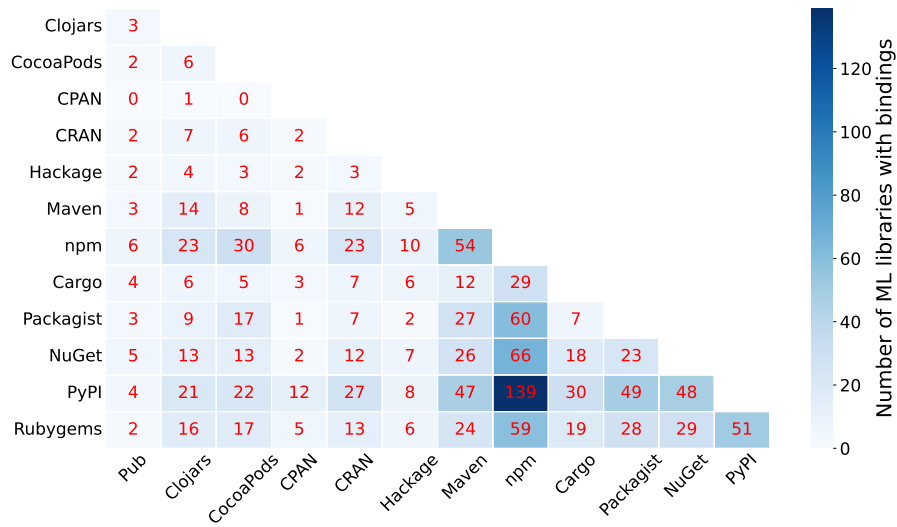
	Pub	Clojars	CocoaPods	CPAN	CRAN	Hackage	Maven	npm	Cargo	Packagist	NuGet	PyPI
Clojars	3											
CocoaPods	2	6										
CPAN	0	1	0									
CRAN	2	7	6	2								
Hackage	2	4	3	2	3							
Maven	3	14	8	1	12	5						
npm	6	23	30	6	23	10	54					
Cargo	4	6	5	3	7	6	12	29				
Packagist	3	9	17	1	7	2	27	60	7			
NuGet	5	13	13	2	12	7	26	66	18	23		
PyPI	4	21	22	12	27	8	47	139	30	49	48	
Rubygems	2	16	17	5	13	6	24	59	19	28	29	51

Number of ML libraries with bindings (color scale: 0 – 120)

Fig. 4: Combinations of software package ecosystems in which ML libraries with bindings are available. The elements represent the number of libraries that can be found in both ecosystems (i.e., ecosystems in the row and column).

libraries with bindings. One reason could be that Python is the most popular language for ML development (Ben Braiek et al., 2018) and JavaScript has been the most commonly used programming language (StackOverflow, 2023). Hence, there could be a need for ML in `npm`, resulting in more support for such bindings. Other notable ecosystem pairs include `npm`-`Packagist`, `Cargo`-`PyPI`, and `npm`-`Maven`, reflecting a diverse landscape of ML library availability and collaboration.

Official organizations behind ML libraries with bindings prefer to focus on a single ecosystem, with PyPI being the most popular choice. Among ML libraries with bindings, 43% (236 out of 546) of them have official packages or officially-maintained bindings. When examining the publication behavior of official organizations behind these 236 ML libraries, we observed that 87% (205 out of 236) of the libraries prefer to focus on a single ecosystem, with `PyPI` being the predominant choice (69%). This preference aligns with prior research by Ben Braiek et al. (2018), which reports Python's dominance in ML development.

> **Takeaway of RQ2**
>
> 55% of ML libraries with bindings are distributed across at least two software package ecosystems, with `npm` being the most popular choice for publishing these bindings. Moreover, the most popular combination of ecosystems to support is `PyPI` and `npm`.

5.3 RQ3: How are cross-ecosystem bindings for popular ML libraries maintained?

Motivation. The popularity of certain ML libraries has led to increased development of bindings across software package ecosystems. However, the development and maintenance practices for these bindings can be different from their host libraries. If a binding only supports a small number of the releases of its host library and has a high delay in getting an update, developers relying on this binding may be forced to use outdated versions for extended periods. This situation poses risks such as exposure to bugs and vulnerabilities present in older versions. Without understanding the maintenance quality of these bindings, developers face challenges regarding the integration of bindings. The findings from this RQ will (1) help developers evaluate the maintenance quality of bindings for ML libraries when selecting bindings, (2) open new research opportunities to explore factors affecting the maintenance of these bindings, (3) guide ML package owners in improving support and communication strategies for bindings, and (4) motive tool builders to create tools to support the maintenance of these bindings.

Approach. For libraries that primarily distribute their packages within a single software package ecosystem or designate a specific package as their main release channel, we considered the versions of these packages as the library's releases. For instance, *PyTorch* adopts a "Python First" approach (PyTorch, 2022), positioning its *torch* package in `PyPI` not merely as a binding but as the main package. On the other hand, for libraries like *OpenPose*, which do not distribute packages through any specific package ecosystem,[11] we relied on source code repository tags as their releases. This method also applies to bindings that span multiple ecosystems, particularly when no main package is declared. In these cases, the source code repository tags often represent the most reliable source of library releases, such as the tags for *XGBoost*.[12]

We assume a one-to-one mapping between the binding and the host library versions. Bindings typically follow one of the following approaches: (1) include the source code of the corresponding host library version, (2) include the pre-built binary of the corresponding host library version, or (3) download and compile the corresponding host library version during installation. For instance, the XGBoost binding package in CRAN[13] follows the first approach by including the source code of a corresponding version of XGBoost. OpenCV's Python binding[14] follows the second approach by including the built OpenCV binaries for the corresponding version. Finally, TensorFlow's Rust binding[15] follows the third approach by automatically downloading or compiling a corresponding version of TensorFlow. Furthermore, some bindings

---

[11] https://github.com/CMU-Perceptual-Computing-Lab/openpose/issues/1250

[12] https://github.com/dmlc/xgboost

[13] https://cran.r-project.org/web/packages/xgboost/index.html

[14] https://github.com/opencv/opencv-python

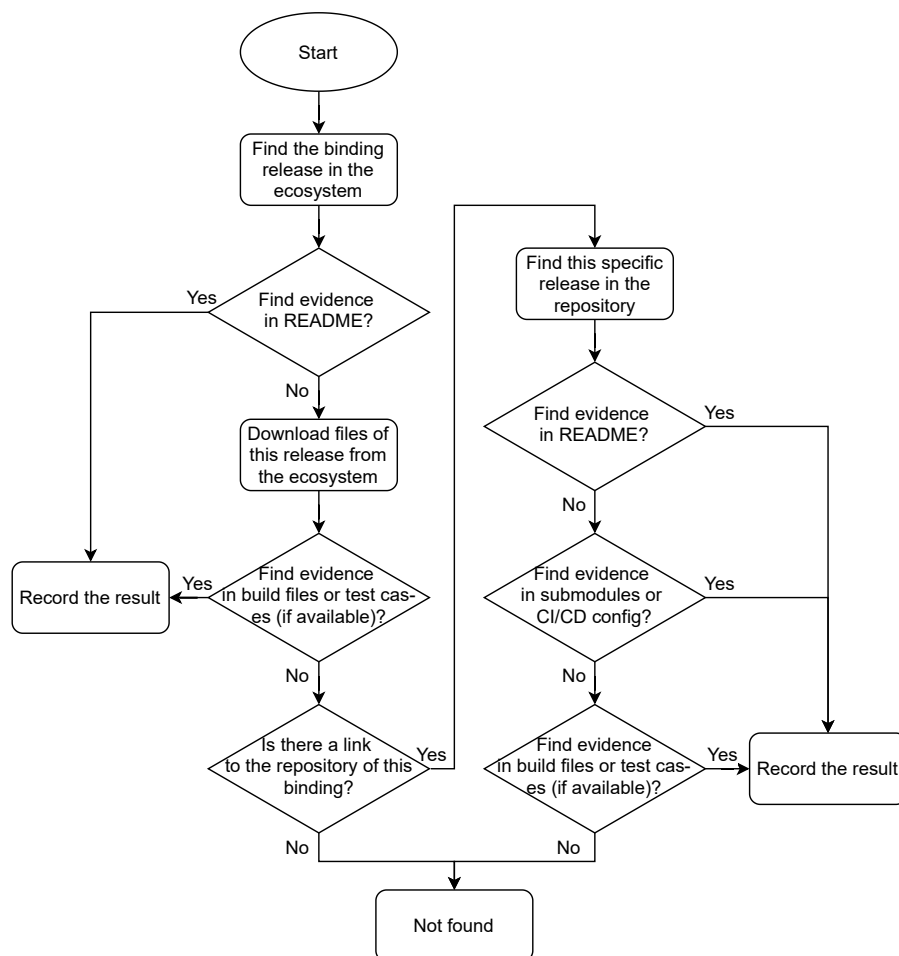[15] https://github.com/tensorflow/rust

Fig. 5: The process of identifying which version of the host library is supported by a specific binding version.

support developers in installing a specific version of the host library locally and specifying this version when building or installing the binding. Subsequently, we matched all releases of each binding with the corresponding host library releases by searching for evidence of this matching in the following places of the binding releases:

1. **README:** When the supported version of the host library is mentioned explicitly.
2. **Git Submodules (Chacon and Straub, 2014):** When the source code of the supported version of the host library is included as a submodule.

3. **Build Files:** When the supported version of the host library that is going to be built for developers is mentioned explicitly, e.g., a binding might indicate the supported version in *CMakeLists.txt* or *Rakefile*.
4. **Test Cases:** When the supported version of the host library is verified explicitly by the tests.
5. **Configurations of continuous integration or delivery (CI/CD):** When the supported version of the host library is indicated explicitly in the configuration files, such as *.travis.yml*, to set up the CI/CD environment.

Figure 5 shows an overview of the identification process. For each release of a cross-ecosystem binding, we inspected the README and downloaded the files of this release from the software package ecosystem. If no evidence could be found, we checked out the source code repository of this binding. To locate the corresponding checkpoint (i.e., a Git commit) of a specific release in the repository, we investigated all tags of the repository and extracted the one that had the same version as the release of the binding. However, some repositories do not have tags for published releases. In this case, we tracked the modification history of the meta-data file (which stores the version number) to locate the checkpoint. Some examples of meta-data files are *package.json* in `npm`, *setup.py* in `PyPI`, and *pom.xml* in `Maven`. After locating the checkpoint of the specific release, we inspected the files to find out which release of the host library is supported. We performed this process for all releases of cross-ecosystem bindings and matched them with the releases of the host library. The matching results can be found in our replication package (Li and Bezemer, 2024).

To quantitatively assess the alignment between binding and host library releases, we extracted the **delay** between the matched releases and the host releases in days and calculated the **coverage** for each binding b in software package ecosystem e as follows:

$$coverage_{e,b} = \frac{\#matched\_host\_releases_{e,b}}{\#release_{host,b}} \qquad (5)$$

where the numerator is the number of host releases that are supported by b and the denominator is the number of releases of the host library. We only consider the releases of the host library that were published since the binding started to provide support. The coverage of a binding b will be 100% if we can find a matched binding release for every host release. If we cannot find any matched release in b, the coverage will be 0%. The coverage metric captures the overall support that a binding offers for an ML library. Figure 6 presents three matching results, the denominators are 3 for all three examples and the numerators are 3, 2, and 0 respectively. Hence, the coverage values for these examples are 100%, 67%, and 0%. In addition, we investigated the **technical lag** (Stringer et al., 2020; Zerouali et al., 2019) of the latest release of each binding. Technical lag occurs when a binding does not support the most recent version of its host library. For example, a major version lag exists when the
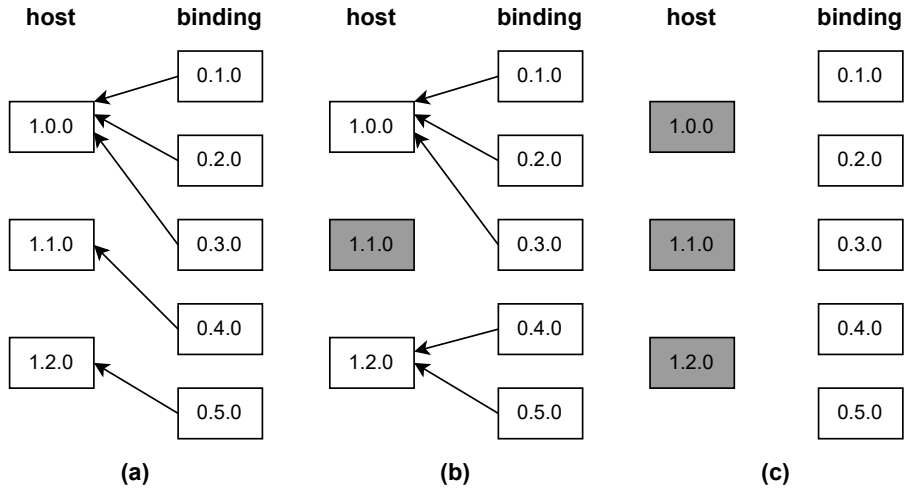
Fig. 6: Three examples of matching binding releases and host releases: (a) all host releases are supported by the binding; (b) 2 out of 3 host releases are supported; (c) no host releases are supported.

latest version of a binding supports version *1.1.0* of its host library, but the latest available version of the host library is *2.1.0*. Similarly, minor and micro version lags exist when there are discrepancies in the minor or micro version numbers respectively.

Next, we compared our findings between officially-maintained bindings and community-maintained bindings by performing the Mann-Whitney U test (Mann and Whitney, 1947) at a significance level of $\alpha = 0.05$ to determine whether the differences are significant. Also, we computed Cliff's delta d (Long et al., 2003) effect size to quantify the difference. To explain the value of d, we use the thresholds which are provided by Romano et al. (2006):

$$\text{Effect size} = \begin{cases} negligible, & \text{if } |d| \leq 0.147 \\ small, & \text{if } 0.147 < |d| \leq 0.33 \\ medium, & \text{if } 0.33 < |d| \leq 0.474 \\ large, & \text{if } 0.474 < |d| \leq 1 \end{cases} \tag{6}$$

Findings. **While npm is the most popular ecosystem for cross-ecosystem bindings of popular ML libraries, developers in PyPI are more likely to find a matched release of a cross-ecosystem binding for popular ML host libraries.** We found that 48% of the popular ML libraries have bindings in `npm` compared to 33% in `PyPI`. However, Figure 7 shows that `PyPI` has the highest coverage among the 13 studied ecosystems, with a median value of 46%. In contrast, other ecosystems have relatively small coverage values. We observed that the main reason for the low coverage of these bindings is that they only provide support for a specific subset of the releases from
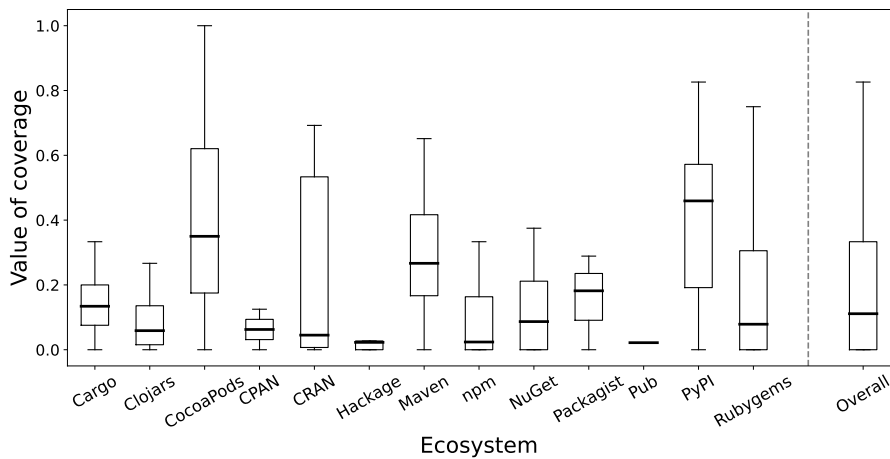
Fig. 7: The distributions of the coverage of cross-ecosystem bindings for popular ML libraries across ecosystems.

their host libraries. For example, a binding of *dlib* in `RubyGems` has 13 releases but only supports 2 out of 50 versions of the host library. Specifically, releases *1.0.0* to *1.0.3* of this binding support version *v18.13* of the host library, then it skipped nine versions (i.e., *v18.14* to *v19.3*) of the host library and published 9 releases to support *v19.4*. This phenomenon can also be found in other bindings with low coverage. In addition, we noticed that the median coverage values of `CRAN`, `Hackage`, `npm`, and `Pub` are below 5%. This phenomenon may be attributed to a large portion of community-maintained bindings within these ecosystems (e.g., 84% in `npm`), where many bindings stick with a single host library version.

After a release of the ML library was published, their bindings in PyPI and Maven received a corresponding update more quickly than the ones in other software package ecosystems. Figure 8 shows that bindings within the `PyPI` and `Maven` ecosystems tend to be updated to match new releases of ML libraries more swiftly than those in other ecosystems, with median delays of less than 7 days. In contrast, bindings in ecosystems like `Packagist` and `Hackage` may experience median delays exceeding 365 days, indicating a slower pace of alignment with host library updates.

Compared to community-maintained bindings for popular ML libraries, the officially-maintained bindings have higher coverage, shorter delays, and smaller technical lags. Among the 133 bindings for the 40 popular ML libraries, 36 are officially-maintained bindings while 97 are community-maintained bindings (27% versus 73%). Figure 9 shows that the coverage of community-maintained bindings is mostly below 0.2 and the delays with the host libraries are mostly between 29 and 275 days. In contrast, the officially-maintained bindings have much more coverage (median of 11%) and less delay (median of 34 days). The Mann-Whitney U test demonstrates sig-
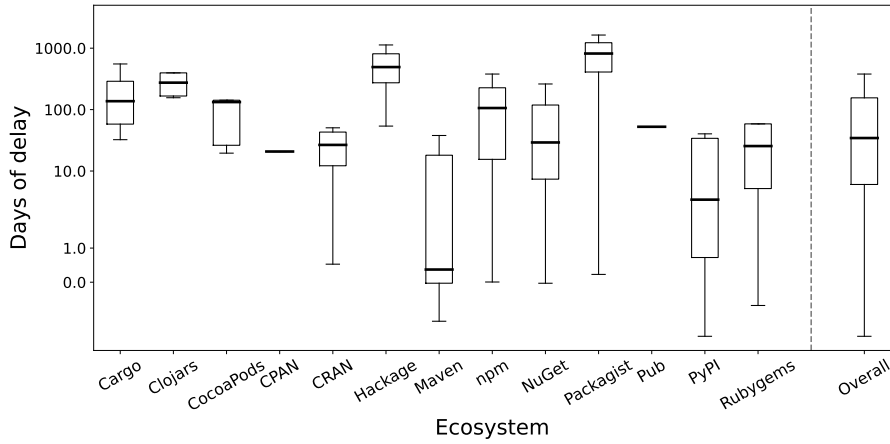
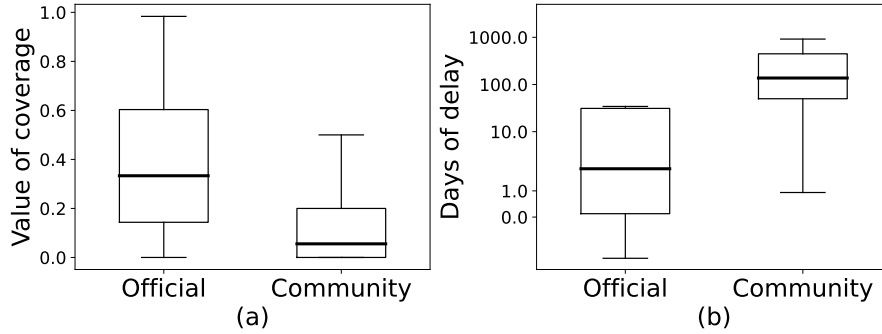Fig. 8: The distributions of the delay between releases of popular ML host libraries and their bindings.

Fig. 9: Comparisons of the cross-ecosystem bindings for ML libraries which are maintained by the official organization and the community: (a) the distributions of the coverage; (b) the distributions of the delay between a release of the host library and the corresponding release of the binding.

nificant differences in both coverage and delay distributions between officially-maintained bindings and community-maintained bindings. In addition, the values of Cliff's Delta d are 0.475 and -0.819 respectively, indicating large effect sizes. Furthermore, 45% of the officially-maintained bindings experience zero technical lag, compared to 24% of community-maintained bindings. Although officially-maintained bindings show lower proportions of major (21% vs. 32%) and minor (27% vs. 40%) technical lags compared to community-maintained bindings, they have a slightly higher proportion of micro lags (7% vs. 4%).

69% of the cross-ecosystem bindings do not follow the version numbers of their host library and 29% of cross-ecosystem bindings do not specify the matching information anywhere. During the process of matching binding releases with their host releases, we noticed that the majority (92 out of 133) of bindings do not reuse any version number from their host libraries. This misalignment in version numbers could lead to confusion for developers. Furthermore, our manual review process revealed that 29% (38 out of 133) of the bindings do not provide explicit matching information. Particularly, the absence of matching information in community-maintained bindings (89% of those are without matching information) creates an additional challenge, potentially forcing developers to resort to trial-and-error methods to identify compatible versions.

> **Takeaway of RQ3**
>
> Generally, the release coverage of cross-ecosystem bindings for their host library is low and the delay is large, with PyPI and Maven performing better than other ecosystems. Officially-maintained bindings for popular ML libraries offer significantly better coverage, faster updates, and less technical lag than community-maintained alternatives. However, 69% of bindings do not follow the version numbers of their host libraries and 29% of bindings lack explicit version matching information, posing challenges for developers.

6 Implications

In this section, we discuss the implications of our findings for developers, owners of ML host libraries and their cross-ecosystem bindings, researchers, tool builders, and repository hosting platforms.

6.1 Implications for Developers

Developers are not always limited to using the same source programming language as a popular ML library when they wish to use this library. PyPI is the most popular choice for ML libraries (Ben Braiek et al., 2018) and developers in PyPI are more likely to find an ML library for their own projects. However, developers might prefer to start a project in their most familiar ecosystem. Our findings show that 5% of the ML repositories are libraries that can be found in multiple ecosystems (Secion 5.1) and their bindings spread across different ecosystems, with npm being the most popular choice for bindings (Secion 5.2). Therefore, developers may find a binding of their desired ML library in the chosen software package ecosystem. We suggest that developers should put the choice of ecosystems into consideration before starting a project.

Developers should consider the number of supported releases and the delay of getting a corresponding update when choosing the binding for an ML library. Usually, cross-ecosystem bindings for popular ML libraries do not support all releases of their host library (Section 5.3). Our findings show that it is not sufficient to look at the number of binding releases only. Bindings with low coverage could publish many releases but only support one or two versions of their host library. If developers are going to adopt such a binding, they should consider that it might not support a needed version in the future. For a binding with high coverage, developers should also consider how long it takes to update for a version of their host library and whether such delays are acceptable to them. By checking the maintenance history of the chosen binding, developers can have an expectation about the binding and consider whether they wish to adopt it or not.

6.2 Implications for ML Package Owners

Owners of cross-ecosystem bindings for popular ML libraries should explicitly indicate the matching between releases of the binding and releases of the host library. In Section 5.3, we found that some software package ecosystems have median coverage rates below 5% for bindings of popular ML host libraries. Such a low coverage indicates that either a small portion of the host releases are supported, or it is not possible to find out which versions are supported (even after our thorough investigation). In addition, we noticed that only 31% of cross-ecosystem bindings follow the version numbers of their host library. We recommend that owners of cross-ecosystem bindings use the same version number as their host libraries and indicate that in their README. For example, the binding of *mlpack* in `CRAN` mentions that "the version number of MLPACK is used as the version number of this package" (Eddelbuettel and Balamuta, 2020). Also, we recommend adding an extra number after the original version number, for example, changing the version number from "1.2.3" to "1.2.3.0". This way, the owner can increase the extra number when fixing bugs in the binding without causing confusion for developers.

Owners of popular ML host libraries should take notice of the c ommunity-maintained bindings for their libraries. 73% of the cross-ecosystem bindings for popular ML libraries are maintained by the community (Section 5.3). These community-maintained bindings help developers to use the functionalities from their host library in an ecosystem which the official organization does not support. We recommend that official organizations keep an eye on the community-maintained bindings. For example, the official organization could inform popular community-maintained bindings about important updates, e.g., those that fix security vulnerabilities. We noticed that some libraries list the community-maintained bindings in their README or the official website. Furthermore, we observed that *OpenCV* even adopted a popular community-maintained binding (OpenCV, 2021).

6.3 Implications for Researchers

Researchers should expand research into bindings across software domains. The efficacy of `BindFind` in identifying and analyzing bindings (Section 5.1) underscores a rich avenue for further exploration. We suggest researchers apply `BindFind` in broader contexts, extending its use to examine areas such as web development frameworks. This extension could provide valuable insights into how bindings enhance software library interoperability across various domains. Researchers can reuse our replication package (Li and Bezemer, 2024) which contains a dataset of 250,668 bindings and their host names identified by `BindFind`.

Researchers should further investigate the differences between offic ially-maintained bindings and community-maintained bindings. Our results show that community-maintained bindings and officially-maintained bindings have different coverages, delays, and technical lags (Section 5.3). Future studies should investigate what causes these differences. One factor could be simply financial incentives (e.g., because contributors to officially maintained bindings work for the company driving the binding), but there could also be socio-technical factors. For example, developers may be more motivated to contribute to officially-maintained bindings as such contributions are considered more valuable or prestigious. In addition, future studies should investigate how the communication between developers of community-maintained bindings and the host library can be improved.

Researchers should study automatic matching tools for releases of ecosystem bindings to match with releases of their host library. It is a complex and tedious task to identify which host library release is supported by an ecosystem binding. Automatic version matching tools for ecosystem bindings can help developers to find a suitable release without going through all the related files of a binding (like we did in Section 5.3) or trying the releases one by one in their project.

Researchers should study the impact of using bindings on ML development. Our replication package (Li and Bezemer, 2024) provides information on 546 ML libraries and their bindings. Researchers can utilize this dataset to investigate various aspects of ML development. For instance, prior work (Li et al., 2024) studied the impact of bindings for *TensorFlow* and *PyTorch* on software quality in terms of correctness and time cost. Future research can explore how bindings for other ML libraries affect different aspects of software development, such as reliability, maintainability, and efficiency.

Researchers should explore the adoption of bindings for ML libraries across software ecosystems. The analysis of the ecosystem distribution of bindings for ML libraries reveals popular ecosystems and ecosystem-pairs (Section 5.2). These findings suggest a need for further research into the adoption of these bindings across different programming languages and ecosystems. For example, LinkedIn's adoption of TensorFlow's JavaScript bindings over Python

and Java highlights how ecosystem choices can influence the deployment and integration of ML models within existing infrastructure.[16] By studying such cases, researchers can gain deeper insights into the factors that drive ecosystem preferences and how these decisions affect the success of software projects. Researchers can leverage the dataset provided in our replication package (Li and Bezemer, 2024), which includes detailed information about the distribution of bindings across ecosystems, to conduct their analyses.

6.4 Implications for Tool Builders

Tool builders should create tools to automate and simplify the maintenance of bindings for ML libraries. Maintaining cross-ecosystem bindings is a challenging task, particularly for community-maintained bindings, indicated by their often lower coverage and higher delays compared to officially-maintained bindings (Section 5.3). Tool builders should develop tools to monitor new releases of a host library, automatically suggest updates for the bindings to reduce the delay in supporting the latest versions, and check the synchronization of version numbers between bindings and their host libraries. Integrating these tools with CI/CD pipelines can help identify compatibility issues early, improve coverage, and reduce delays for both community and officially maintained bindings.

6.5 Implications for Repository Hosting Platforms

Repository hosting platforms should integrate `BindFind` to discover bindings. As demonstrated in Section 5.1, `BindFind` can accurately identify bindings and extract their host names, making it a valuable tool for repository hosting platforms. For instance, the Tesseract OCR project maintains a manual list of its bindings in the documentation.[17] By integrating `BindFind`, repository hosting platforms such as GitHub can automate the discovery of bindings for various host libraries, providing a list of available bindings for developers.

7 Threats to Validity

In this section, we discuss the threats to validity of our study about popular ML libraries and their cross-ecosystem bindings.

---

[16] `https://blog.tensorflow.org/2022/03/how-linkedin-personalized-performance.html`

[17] `https://tesseract-ocr.github.io/tessdoc/AddOns.html#tesseract-wrappers`

7.1 Internal Validity

We use release coverage as a metric to reason about the maintenance of a binding. A low coverage value does not always mean that a binding has a bad support. For example, a binding can be forward-compatible, thereby making it easier to support newer versions without changing the binding. However, developers have to manually verify whether this binding will work for a specific version of the ML library that they are going to use, as it is not indicated anywhere. A binding can also be backward-compatible, thereby making it support older versions of the host library. However, it is not guaranteed in many cases, as bindings do not typically undergo testing on older versions of the host library, and developers have to verify this themselves.

We assume a one-to-one mapping when matching the release versions in bindings to the ML host libraries. This assumption is based on common practices where bindings either include the exact source code, include pre-built binaries, or download and compile the matching version. However, this assumption does not account for scenarios where a binding supports multiple versions of the host library within a single version.

We classified cross-ecosystem bindings of popular ML libraries into officially-maintained bindings and community-maintained bindings. However, some community-maintained bindings could be adopted by the official organization and become an officially-maintained binding. In our study, we only consider the latest information and do not take into account the history of ownership. In addition, the owner of a community-maintained binding could be a member of the official organization of the host library. If the binding is not owned by the official organization and there is no evidence in its README, we consider such bindings as community-maintained bindings.

Some software package ecosystems like `PyPI` allow the owner of a package to delete a published release. `Libraries.io` does not record the history of releases of a package, hence, we only consider the information of releases when the dataset was collected. In addition, some ecosystems also support the owner of a package to deprecate a release instead of deleting a release, e.g., `npm` and `Cargo`. In our study, we consider all releases which also include the deprecated ones.

7.2 External Validity

In our empirical study, we studied cross-ecosystem bindings for ML libraries. The results of our study might not apply directly to all cross-ecosystem bindings. Still, our methodology can be applied to analyze other cross-ecosystem bindings for libraries in other domains. In addition, we focused on open source ML libraries as many popular ML libraries are open source. Future studies should investigate if our findings hold for proprietary ML libraries.

8 Conclusion

In this study, we introduced `BindFind`, a novel approach for identifying bindings and extracting host names within software package ecosystems. Applying `BindFind` to the `libraries.io` dataset allowed us to gather 2,436 bindings for 546 ML libraries across 13 software package ecosystems. We analyzed the population of bindings for ML libraries and the distribution of these libraries and their bindings across various ecosystems. In addition, we identified 40 popular ML libraries along with their 133 cross-ecosystem bindings, further examining their releases. Based on the data from these popular ML libraries, we conducted an in-depth analysis to investigate the development and maintenance of these cross-ecosystem bindings. We shared the collected data in our replication package (Li and Bezemer, 2024). The most important findings of our study are:

1. `BindFind` demonstrates high F1 scores in identifying bindings and extracting host names, indicating its robust capability to assist in the discovery of bindings within software package ecosystems.
2. `npm` is the most popular software package ecosystem for bindings of ML libraries, with `npm` and `PyPI` emerging as the predominant combination for publishing these bindings.
3. The study highlights significant challenges in the maintenance of cross-ecosystem bindings, including limited release coverage, delays in updating bindings to match new releases, and prevalent technical lag, especially among community-maintained bindings.
4. Most bindings do not follow their host library's version numbers, and many lack clear information on which host library versions they support.

Our findings show that developers who wish to use a popular ML library are not limited to using the programming language the library was written in, as there exist many cross-ecosystem bindings. However, they should carefully check the coverage, delay, and technical lag of these bindings before they commit to using one. In addition, we suggest that maintainers of cross-ecosystem bindings should follow the version number of their host library and add an extra number after it, to account for bug fixes, and make it easier for developers to identify which version of the host library is supported by the binding.

Data Availability Statement

The data generated and analyzed during this study is available in our replication package at `https://doi.org/10.5281/zenodo.12746638`.

References

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X (2016) TensorFlow: A system for large-scale machine learning. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, USENIX Association, USA, OSDI'16, pp 265–283, DOI 10.5555/3026877.3026899

Barros-Justo JL, Pinciroli F, Matalonga S, Martínez-Araujo N (2018) What software reuse benefits have been transferred to the industry? a systematic mapping study. Information and Software Technology 103:1–21

Bavota G, Canfora G, Penta MD, Oliveto R, Panichella S (2013) The evolution of project inter-dependencies in a software ecosystem: The case of Apache. In: 2013 IEEE International Conference on Software Maintenance, IEEE Computer Society, USA, ICSM '13, pp 280–289, DOI 10.1109/ICSM.2013.39

Ben Braiek H, Khomh F, Adams B (2018) The Open-Closed Principle of Modern Machine Learning Frameworks. In: Proceedings of the 15th International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, MSR '18, pp 353–363, DOI 10.1145/3196398.3196445

Bernardo JH, Da Costa DA, Medeiros SQd, Kulesza U (2024) How do machine learning projects use continuous integration practices? an empirical study on GitHub actions. In: Proceedings of the 21st International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, MSR '24, pp 665–676, DOI 10.1145/3643991.3644915

Blincoe K, Harrison F, Damian D (2015) Ecosystems in GitHub and a method for ecosystem identification using reference coupling. In: Proceedings of the 12th Working Conference on Mining Software Repositories, IEEE Press, MSR '15, pp 202–207, DOI 10.5555/2820518.2820544

Blincoe K, Harrison F, Kaur N, Damian D (2019) Reference coupling: An exploration of inter-project technical dependencies and their characteristics within large software ecosystems. Information and Software Technology 110:174–189, DOI 10.1016/j.infsof.2019.03.005

Bogart C, Kästner C, Herbsleb J, Thung F (2016) How to Break an API: Cost Negotiation and Community Values in Three Software Ecosystems. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, FSE 2016, pp 109–120, DOI 10.1145/2950290.2950325

Bogart C, Kästner C, Herbsleb J, Thung F (2021) When and how to make breaking changes: Policies and practices in 18 open source software ecosystems. ACM Transactions on Software Engineering and Methodology (TOSEM) 30(4):1–56, DOI 10.1145/3447245

Borges H, Hora A, Valente MT (2016) Understanding the factors that impact the popularity of GitHub repositories. In: 2016 IEEE International

Conference on Software Maintenance and Evolution (ICSME), pp 334–344, DOI 10.1109/ICSME.2016.31

Chacon S, Straub B (2014) Pro Git, 2nd edn. Apress, New York, NY, DOI 10.1007/978-1-4842-0076-6

Chiba S (2019) Foreign language interfaces by code migration. In: Proceedings of the 18th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, Association for Computing Machinery, New York, NY, USA, GPCE 2019, pp 1–13, DOI 10.1145/3357765.3359521

Cogo FR, Oliva GA, Hassan AE (2021) An empirical study of dependency downgrades in the npm ecosystem. IEEE Transactions on Software Engineering 47(11):2457–2470, DOI 10.1109/TSE.2019.2952130

Constantinou E, Mens T (2017a) An empirical comparison of developer retention in the RubyGems and npm software ecosystems. Innovations in Systems and Software Engineering 13(2):101–115, DOI 10.1007/s11334-017-0303-4

Constantinou E, Mens T (2017b) Socio-technical evolution of the Ruby ecosystem in GitHub. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 34–44, DOI 10.1109/SANER.2017.7884607

Constantinou E, Decan A, Mens T (2018) Breaking the borders: An investigation of cross-ecosystem software packages. In: Gousios G, Hejderup J (eds) Proceedings of the 17th Belgium-Netherlands Software Evolution Workshop, Delft, the Netherlands, December 10th - to - 11th, 2018, CEUR-WS.org, CEUR Workshop Proceedings, vol 2361, pp 1–5

Decan A, Mens T (2020) How Magic Is Zero? An Empirical Analysis of Initial Development Releases in Three Software Package Distributions. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, Association for Computing Machinery, New York, NY, USA, ICSEW'20, pp 695–702, DOI 10.1145/3387940.3392205

Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Burstein J, Doran C, Solorio T (eds) Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, pp 4171–4186, DOI 10.18653/v1/N19-1423

Dilhara M, Ketkar A, Dig D (2021) Understanding software-2.0: A study of machine learning library usage and evolution. ACM Trans Softw Eng Methodol 30(4):1–42, DOI 10.1145/3453478

Eddelbuettel D, Balamuta JJ (2020) Rcppmlpack. URL `https://github.com/rcppmlpack/RcppMLPACK1`, Accessed: 13 March 2024

Fang H, Lamba H, Herbsleb J, Vasilescu B (2022) "this is damn slick!": estimating the impact of tweets on open source project popularity and new contributors. In: Proceedings of the 44th International Conference on Software Engineering, Association for Computing Machinery, New York, NY, USA, ICSE '22, pp 2116–2129, DOI 10.1145/3510003.3510121

Finne S, Leijen D, Meijer E, Peyton Jones S (1998) H/Direct: A binary foreign language interface for Haskell. In: Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming, Association for Computing Machinery, New York, NY, USA, ICFP '98, pp 153–162, DOI 10.1145/289423.289437

Franco-Bedoya O, Ameller D, Costal D, Franch X (2017) Open source software ecosystems: A systematic mapping. Information and Software Technology 91:160–185, DOI https://doi.org/10.1016/j.infsof.2017.07.007

Furr M, Foster JS (2008) Checking type safety of foreign function calls. ACM Trans Program Lang Syst 30(4):18:1–18:63, DOI 10.1145/1377492.1377493

Gao K, He R, Xie B, Zhou M (2024) Characterizing deep learning package supply chains in pypi: Domains, clusters, and disengagement. ACM Transactions on Software Engineering and Methodology 33(4):1–27, DOI 10.1145/3640336, URL `http://dx.doi.org/10.1145/3640336`

German DM, Adams B, Hassan AE (2013) The Evolution of the R Software Ecosystem. In: 17th European Conference on Software Maintenance and Reengineering, IEEE Press, CSMR, pp 243–252, DOI 10.1109/CSMR.2013.33

Godbole S, Sarawagi S (2004) Discriminative methods for multi-labeled classification. In: Pacific-Asia conference on knowledge discovery and data mining, Springer, pp 22–30

Grichi M, Eghan EE, Adams B (2020) On the impact of multi-language development in machine learning frameworks. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp 546–556

Guo Q, Chen S, Xie X, Ma L, Hu Q, Liu H, Liu Y, Zhao J, Li X (2019) An Empirical Study Towards Characterizing Deep Learning Development and Deployment Across Different Frameworks and Platforms. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp 810–822, DOI 10.1109/ASE.2019.00080

Han J, Deng S, Xia X, Wang D, Yin J (2019) Characterization and prediction of popular projects on GitHub. In: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), vol 1, pp 21–26, DOI 10.1109/COMPSAC.2019.00013

Han J, Deng S, Lo D, Zhi C, Yin J, Xia X (2020) An Empirical Study of the Dependency Networks of Deep Learning Libraries. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE Press, pp 868–878, DOI 10.1109/ICSME46990.2020.00116, iSSN: 2576-3148

Heinemann L, Deissenboeck F, Gleirscher M, Hummel B, Irlbeck M (2011) On the extent and nature of software reuse in open source java projects. In: Proceedings of the 12th International Conference on Top Productivity through Software Reuse, Springer-Verlag, Berlin, Heidelberg, ICSR'11, p 207–222

International Organization for Standardization (2016) ISO/IEC TR 10182:2016, Information technology - Programming languages, their environments and system software interfaces - Guidelines for language

bindings. International Standard, URL `https://www.iso.org/standard/67465.html`

Islam MJ, Nguyen HA, Pan R, Rajan H (2019) What Do Developers Ask About ML Libraries? A Large-scale Study Using Stack Overflow. `1906.11940`

Jansen S (2014) Measuring the health of open source software ecosystems: Beyond the scope of project health. Information and Software Technology 56(11):1508–1519, DOI https://doi.org/10.1016/j.infsof.2014.04.006, special issue on Software Ecosystems

Kannee K, Kula R, Wattanakriengkrai S, Matsumoto K (2023) Intertwining communities: Exploring libraries that cross software ecosystems. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), IEEE Computer Society, Los Alamitos, CA, USA, pp 518–522, DOI 10.1109/MSR59073.2023.00077

Katz J (2020) Libraries.io Open Source Repository and Dependency Metadata. DOI 10.5281/zenodo.3626071, URL `https://doi.org/10.5281/zenodo.3626071`

Kochanthara S, Dajsuren Y, Cleophas L, van den Brand M (2022) Painting the landscape of automotive software in GitHub. In: 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR), pp 215–226, DOI 10.1145/3524842.3528460

Kula RG, Robles G (2019) The Life and Death of Software Ecosystems, Springer Singapore, Singapore, pp 97–105. DOI 10.1007/978-981-13-7099-1_6

Kula RG, Roover CD, Germán DM, Ishio T, Inoue K (2018) A generalized model for visualizing library popularity, adoption, and diffusion within a software ecosystem. In: 25th International Conference on Software Analysis, Evolution and Reengineering, IEEE Computer Society, SANER 2018, pp 288–299, DOI 10.1109/SANER.2018.8330217

Lan Z, Chen M, Goodman S, Gimpel K, Sharma P, Soricut R (2020) ALBERT: A lite BERT for self-supervised learning of language representations. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, pp 1–17, URL `https://openreview.net/forum?id=H1eA7AEtvS`

Lee B, Wiedermann B, Hirzel M, Grimm R, McKinley KS (2010) Jinn: synthesizing dynamic bug detectors for foreign language interfaces. SIGPLAN Not 45(6):36–49, DOI 10.1145/1809028.1806601

Lei H, Zhang S, Wang J, Xiao G, Liu Y, Sui Y (2023) Why do deep learning projects differ in compatible framework versions? an exploratory study. In: 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), pp 509–520, DOI 10.1109/ISSRE59848.2023.00076

Li H, Bezemer CP (2024) Bridging the Language Gap: An Empirical Study of Bindings for Open Source Machine Learning Libraries in Software Package Ecosystems. DOI 10.5281/zenodo.12746638, URL `https://doi.org/10.5281/zenodo.12746638`

Li H, Cogo FR, Bezemer CP (2023) An empirical study of yanked releases in the rust package registry. IEEE Transactions on Software Engineering 49(1):437–449, DOI 10.1109/TSE.2022.3152148

Li H, Rajbahadur GK, Bezemer CP (2024) Studying the impact of tensorflow and pytorch bindings on machine learning software quality. ACM Trans Softw Eng Methodol pp 1–31, DOI 10.1145/3678168, URL `https://doi.org/10.1145/3678168`, just Accepted

Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, Stoyanov V (2019) RoBERTa: A robustly optimized bert pretraining approach

Long JD, Feng D, Cliff N (2003) Ordinal Analysis of Behavioral Data, American Cancer Society, chap 25, pp 635–661. DOI 10.1002/0471264385.wei0225

Manikas K (2016) Revisiting software ecosystems research: A longitudinal literature study. Journal of Systems and Software 117:84–103, DOI https://doi.org/10.1016/j.jss.2016.02.003

Mann HB, Whitney DR (1947) On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. The Annals of Mathematical Statistics 18(1):50–60, DOI 10.1214/aoms/1177730491

Mens T, Claes M, Grosjean P, Serebrenik A (2014) Studying Evolving Software Ecosystems based on Ecological Models, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 297–326. DOI 10.1007/978-3-642-45398-4_10

Mohagheghi P, Conradi R (2007) Quality, productivity and economic benefits of software reuse: a review of industrial studies. Empirical Software Engineering 12:471–516

Nakata S, Sugaya M, Kuramitsu K (2011) Fault model of foreign function interface across different domains. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W), pp 248–253, DOI 10.1109/DSNW.2011.5958850, iSSN: 2325-6664

OBrien D, Biswas S, Imtiaz S, Abdalkareem R, Shihab E, Rajan H (2022) 23 shades of self-admitted technical debt: an empirical study on machine learning software. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, ESEC/FSE 2022, pp 734–746, DOI 10.1145/3540250.3549088

OpenCV (2021) OpenCV-Python is now an official OpenCV project. URL `https://opencv.org/blog/opencv-python-is-now-an-official-opencv-project`, Accessed: 13 March 2024

Osman K, Baysal O (2021) Health is wealth: Evaluating the health of the bitcoin ecosystem in GitHub. In: 2021 IEEE/ACM 4th International Workshop on Software Health in Projects, Ecosystems and Communities (SoHeal), pp 1–8, DOI 10.1109/SoHeal52568.2021.00007

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) PyTorch: An imperative style, high-performance deep learning library. In:

Advances in Neural Information Processing Systems 32, Curran Associates, Inc., NeurIPS, pp 8024–8035

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-Learn: Machine learning in Python. the Journal of machine Learning research 12:2825–2830

PyTorch (2021) PyTorch Mobile, end-to-end workflow from training to deployment for ios and android mobile devices. URL `https://pytorch.org/mobile/ios`, Accessed: 13 March 2024

PyTorch (2022) Pytorch design philosophy. URL `https://pytorch.org/docs/stable/community/design.html`, Accessed: 13 March 2024

Rajpurkar P, Zhang J, Lopyrev K, Liang P (2016) SQuAD: 100,000+ questions for machine comprehension of text. In: Su J, Duh K, Carreras X (eds) Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Austin, Texas, pp 2383–2392, DOI 10.18653/v1/D16-1264

Rajpurkar P, Jia R, Liang P (2018) Know what you don't know: Unanswerable questions for SQuAD. In: Gurevych I, Miyao Y (eds) Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Association for Computational Linguistics, Melbourne, Australia, pp 784–789, DOI 10.18653/v1/P18-2124

Reppy J, Song C (2006) Application-specific foreign-interface generation. In: Proceedings of the 5th international conference on Generative programming and component engineering, Association for Computing Machinery, New York, NY, USA, GPCE '06, pp 49–58, DOI 10.1145/1173706.1173714

Romano J, Kromrey JD, Coraggio J, Skowronek J, Devine L (2006) Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's d indices the most appropriate choices. In: annual meeting of the Southern Association for Institutional Research, Citeseer, pp 1–51

Sanh V, Debut L, Chaumond J, Wolf T (2019) DistilBERT, a distilled version of bert: smaller, faster, cheaper and lighter

da Silva Amorim S, McGregor JD, de Almeida ES, von Flach G Chavez C (2016) Software ecosystems architectural health: Challenges x practices. In: Proccedings of the 10th European Conference on Software Architecture Workshops, Association for Computing Machinery, New York, NY, USA, ECSAW '16, DOI 10.1145/2993412.3011881

Smilkov D, Thorat N, Assogba Y, Nicholson C, Kreeger N, Yu P, Cai S, Nielsen E, Soegel D, Bileschi S, Terry M, Yuan A, Zhang K, Gupta S, Sirajuddin S, Sculley D, Monga R, Corrado G, Viegas F, Wattenberg MM (2019) TensorFlow.js: Machine learning for the web and beyond. In: Talwalkar A, Smith V, Zaharia M (eds) Proceedings of Machine Learning and Systems, vol 1, pp 309–321

Song X, Wang Y, Cheng X, Liang G, Wang Q, Zhu Z (2024) Efficiently trimming the fat: Streamlining software dependencies with Java reflection and dependency analysis. In: Proceedings of the IEEE/ACM 46th International

Conference on Software Engineering, Association for Computing Machinery, New York, NY, USA, ICSE '24, pp 1–12, DOI 10.1145/3597503.3639123

de Souza CR, Figueira Filho F, Miranda M, Ferreira RP, Treude C, Singer L (2016) The social side of software platform ecosystems. In: Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, Association for Computing Machinery, New York, NY, USA, pp 3204–3214, DOI 10.1145/2858036.2858431

StackOverflow (2023) Stack Overflow Annual Developer Survey 2023. URL `https://survey.stackoverflow.co/2023`, Accessed: 13 March 2024

Steglich C, Majdenbaum A, Marczak S, Santos R (2020a) A study on organizational it security in mobile software ecosystems literature. In: 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), pp 234–241, DOI 10.1109/ICSA-C50368.2020.00047

Steglich C, Marczak S, Santos R, Mosmann LH, Guerra LP, de Souza C, Filho FF, Perin M (2020b) How do business factors affect developers in mobile software ecosystems? In: XVI Brazilian Symposium on Information Systems, Association for Computing Machinery, New York, NY, USA, SBSI'20, pp 1–8, DOI 10.1145/3411564.3411571

Stringer J, Tahir A, Blincoe K, Dietrich J (2020) Technical lag of dependencies in major package managers. In: 2020 27th Asia-Pacific Software Engineering Conference (APSEC), pp 228–237, DOI 10.1109/APSEC51365.2020.00031

Wei M, Huang Y, Wang J, Shin J, Harzevili NS, Wang S (2022) API recommendation for machine learning libraries: how far are we? In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, ESEC/FSE 2022, pp 370–381, DOI 10.1145/3540250.3549124

Wittern E, Suter P, Rajagopalan S (2016) A look at the dynamics of the JavaScript package ecosystem. In: Proceedings of the 13th International Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, MSR '16, pp 351–361, DOI 10.1145/2901739.2901743

Wolter T, Barcomb A, Riehle D, Harutyunyan N (2023) Open source license inconsistencies on GitHub. ACM Trans Softw Eng Methodol 32(5):1–23, DOI 10.1145/3571852

Xia X, Zhao S, Zhang X, Lou Z, Wang W, Bi F (2023) Understanding the archived projects on GitHub. In: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 13–24, DOI 10.1109/SANER56733.2023.00012

Zaimi A, Ampatzoglou A, Triantafyllidou N, Chatzigeorgiou A, Mavridis A, Chaikalis T, Deligiannis I, Sfetsos P, Stamelos I (2015) An empirical study on the reuse of third-party libraries in open-source software development. In: Proceedings of the 7th Balkan Conference on Informatics Conference, Association for Computing Machinery, New York, NY, USA, BCI '15, pp 1–8, DOI 10.1145/2801081.2801087, URL `https://doi.org/10.1145/2801081.2801087`

Zerouali A, Mens T, Gonzalez-Barahona J, Decan A, Constantinou E, Robles G (2019) A formal framework for measuring technical lag in component repositories — and its application to npm. Journal of Software: Evolution and Process 31(8):e2157, DOI https://doi.org/10.1002/smr.2157