Why Do Deep Learning Projects Differ in Compatible Framework Versions? An Exploratory Study

Huashan Lei*, Shuai Zhang*, Jun Wang*, Guanping Xiao*⋆, Yepang Liu†§, Yulei Sui‡
*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China
†Department of Computer Science and Engineering, Southern University of Science and Technology, China
‡School of Computer Science and Engineering, University of New South Wales, Australia
{leihuashan, shuaizhang, junwang, gpxiao}@nuaa.edu.cn, liuyp1@sustech.edu.cn, y.sui@unsw.edu.au

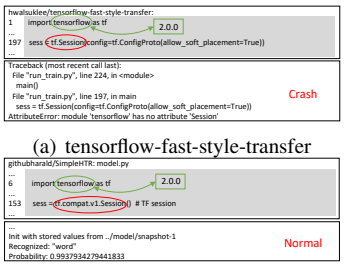
***Abstract*—Deep learning (DL) is becoming increasingly important and widely used in our society. DL projects are mainly built upon DL frameworks, which frequently evolve due to the introduction of new features or bug fixing. Consequently, compatibility issues are commonly seen in DL projects. The compatible framework versions may differ across DL projects, i.e., for a specific framework version, one project runs normally while the other crashes, even if the client code uses the same framework API. Existing studies mainly focus on analyzing the API evolution of Python libraries and the related compatibility issues. However, the difference in framework version compatibility (DFVC) among DL projects has rarely been systematically studied. In this paper, we conduct an empirical study on 90 PyTorch and 50 TensorFlow projects collected from GitHub. By upgrading and downgrading the framework versions, we obtain compatible versions for each project and further investigate the root causes of the different compatible framework versions across projects. We summarize seven root causes: Python version, absence of using the same breaking API, import path, parameter, third-party library, resource, and API usage constraint. We further present six implications based on our empirical findings. Our study can facilitate DL practitioners to gain a better understanding of the DFVC among DL projects.**

***Index Terms*—deep learning, framework version compatibility, empirical study**

I. Introduction

Deep learning (DL) is flourishing in a wide range of our daily lives, such as speech recognition [1], natural language processing [2], robotics [3], and many tasks in software engineering [4]–[8]. DL projects can be easily built upon DL frameworks due to their ease of use, flexibility, and extensive community support [9], [10]. The versions of DL frameworks are frequently changing in fast-paced development. For example, according to the Python-PyPI repository [11], TensorFlow [9] and PyTorch [10], two of the most popular DL frameworks [12], release an average of 13 and 5 versions per year, respectively (as of December 2022).

Such characteristics of DL frameworks have led to prevalent compatibility issues in DL projects. For example, suppose a framework API undergoes a breaking change in a newly released version, i.e., changing the parameter position. In this case, an old API version-based project will inevitably crash after upgrading to the new version [13]. Similarly, a new API version-based project will throw exceptions running in an old framework version environment.



(a) tensorflow-fast-style-transfer

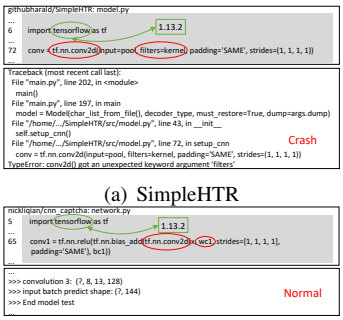
(b) SimpleHTR

Fig. 1. Execution results of tensorflow-fast-style-transfer [14] and SimpleHTR [15] in TensorFlow-2.0.0.

Due to differences in the programming experiences of developers and the configuration of runtime environments, the number of compatible framework versions may differ across DL projects. This means that given a specific framework version, some projects may execute normally while others crash, i.e., ***difference in framework version compatibility (DFVC)*** among DL projects. Studying the underlying causes of DFVC is valuable as it enables us to identify the root issues and develop best practices and guidelines to mitigate compatibility risks associated with specific framework versions. This, in turn, can significantly improve the framework version compatibility of DL projects.

Intuitively, if DL projects employ distinct framework APIs, it is reasonable to expect they could have different compatible framework versions. For example, there are two open-source projects on GitHub, RetinexNet [16] utilizing TensorFlow-1.5.0 and cnn_captcha [17] built upon TensorFlow-1.7.0. When downgrading the TensorFlow version to 1.4.1, RetinexNet crashes at runtime, while cnn_captcha runs normally. By analyzing the traceback message, it is found that RetinexNet calls `reduce_max()`, which is used to compute the maximum of elements across dimensions of a

⋆Guanping Xiao is the corresponding author.
§Yepang Liu is also affiliated with the Research Institute of Trustworthy Autonomous Systems at Southern University of Science and Technology.



(a) SimpleHTR

(b) cnn_captcha

Fig. 2. Execution results of SimpleHTR [15] and cnn_captcha [17] in TensorFlow-1.13.2.

tensor. The `keepdims` parameter of the API was renamed to `keep_dims` in TensorFlow-1.5.0 [18], resulting in the crash of RetinexNet in a TensorFlow-1.4.1 environment. Conversely, cnn_captcha does not use `reduce_max()`, continued to work normally under TensorFlow-1.4.1.

Interestingly, despite utilizing the same APIs, some DL projects still have DFVC. For example, tensorflow-fast-style-transfer [14] crashes when `Session()` is called in TensorFlow-2.0.0, but SimpleHTR [15] works fine when invoking `Session()` in the same framework version, as depicted in Fig. 1. The API is used to manage and execute well-defined operations. Further comparing the client code shows that different import paths of `Session()` cause the DFVC. For tensorflow-fast-style-transfer, `Session()` is imported from `tensorflow` directly, i.e., `tf.Session()`. However, for SimpleHTR, `Session()` is imported from `tf.compat.v1`. Along with the evolution of TensorFlow, several APIs in 1.X versions have been renamed or removed since TensorFlow-2.0 [19]. To support the backward compatibility, TensorFlow-1.X-based projects need to change the API import path, i.e., APIs should be imported from `tensorflow.compat.v1`. Thus, tensorflow-fast-style-transfer will raise the following error during runtime: *AttributeError: module 'tensorflow' has no attribute 'Session'*.

Besides, Fig. 2 shows another two projects that invoke the same API but have DFVC. SimpleHTR crashes when calling 2D convolutional neural network `conv2d()` under TensorFlow-1.13.2. However, cnn_captcha [17], invoking the same `conv2d()`, works fine with this version. By investigating the client code, unlike the example shown in Fig. 1, the DFVC is induced by the difference in the parameter usage of `conv2d()`. SimpleHTR uses the keyword parameter `filters`, while cnn_captcha directly uses the positional parameter. Note that the `filters` parameter was introduced in version 1.14 as an alias for the keyword parameter `filter`, the setting of the convolution layer output dimension [20]. Therefore, invoking `conv2d()` with the keyword parameter `filters` before TensorFlow-1.14 (e.g., 1.13.2), will lead to the following error: *TypeError: conv2d() got an unexpected keyword argument 'filters'*.

On the other hand, if DL projects are developed using different Python versions, they could also have DFVC. For instance, KiU-Net-pytorch [21] was built using Python-3.6. However, the PyPI repository does not provide the installation package for PyTorch-1.11.0. Consequently, attempting to install PyTorch-1.11.0 using the `pip` command in a Python-3.6 environment will result in the failure of PyTorch installation: *ERROR: No matching distribution found for torch==1.11.0*. This means that the project is incompatible with PyTorch-1.11.0. In contrast to KiU-Net-pytorch, SNE-RoadSeg [22] was developed by Python-3.7, which is officially compatible with PyTorch-1.11.0. As a result, the project can be executed successfully in this specific framework version.

From the aforementioned real-world examples, it is evident that even when utilizing the same framework APIs in the client code, DL projects may still have DFVC. However, the occurrence of such phenomena and the underlying causes have received limited research attention. Existing work focuses on analyzing the evolution of Python library APIs [23], [24], automated detecting and repairing deprecated APIs during evolution [25]–[27].

Therefore, it is crucial to study the DFVC of DL projects, which can facilitate DL practitioners to improve the framework version compatibility of client-side code from the lesson learned. To bridge this knowledge gap, in this paper, we conduct a large-scale empirical study on 90 PyTorch-based projects and 50 TensorFlow-based projects collected from GitHub. To test the framework version compatibility of each project, we upgrade/downgrade DL framework versions. Then, we examine the pairs of DL projects with DFVC to investigate the root causes. Our study mainly addresses the following two research questions (RQs):

- **RQ1.** How prevalent is it for DL projects to exhibit differences in compatible framework versions?
- **RQ2.** What are the root causes for the difference in framework version compatibility among DL projects?

Through the examination of RQ1, we can ascertain the prevalence of the DFVC phenomenon in DL projects and conduct a comparative analysis of DFVC between PyTorch and TensorFlow. This RQ guides DL practitioners in selecting the framework with superior version compatibility. By investigating RQ2, we can uncover the fundamental reasons behind DFVC and distill actionable strategies for enhancing the framework version compatibility of DL projects. In this paper, we make the following key contributions:

- To the best of our knowledge, we conduct the first empirical study on the DFVC among DL projects, which could assist DL practitioners in improving framework version compatibility of DL projects.
- We identify seven root causes of the DFVC among DL projects, i.e., Python version, absence of using the same breaking API, import path, parameter, third-party library, resource, and API usage constraint. Our empirical study presents six implications for DL practitioners.
- We make the dataset publicly available at https://doi.org/10.5281/zenodo.8266949.

II. Background

DL projects are projects that employ DL techniques to solve real-world problems. The development of DL projects relies on a large number of software and hardware dependencies. The software and hardware requirements for DL projects vary depending on the specific project and task. We briefly describe some common requirements of software and hardware environments as follows.

Software Dependency. The following are some common software packages that DL projects depend on:

(1) *Operating System:* Windows, macOS, and Linux can all run DL projects. Several practitioners prefer to use the Linux operating system because it is more compatible with production environments [28].

(2) *Programming Language:* Python [29] is the dominant programming language in developing DL projects.

(3) *DL Framework:* DL frameworks provide the key techniques for building neural networks and optimizing techniques. TensorFlow and PyTorch are the two most popular DL frameworks nowadays [12], [30].

(4) *Third-party Library:* Third-party libraries include data processing libraries (e.g., Numpy [31] and Pandas [32]), graphics processing libraries (e.g., OpenCV [33] and Pillow [34]), visualization libraries (e.g., Matplotlib [35] and Seaborn [36]).

(5) *Low-level Library:* CUDA and cuDNN are low-level libraries in DL systems. They provide efficient parallel computing and DL computation acceleration tools that can significantly improve model training and inference speed.

Hardware Dependency. The following are some common hardware that DL projects depend on:

(1) *CPU:* DL projects can run on CPUs, but because DL is computationally intensive, it requires high-performance CPUs [37], such as CPU cluster architectures [38].

(2) *GPU:* GPUs, typically the Nvidia GPUs, are widely used in DL projects since they can perform parallel computing and dramatically increase the speed of training and inference of DL models.

(3) *Memory:* DL projects usually require a lot of memory during the training phase, especially graphics card memory.

III. Methodology

Fig. 3 illustrates the overview of our empirical study. Details of each part are described as follows.

A. Data Collection

Tested DL Projects. The tested DL projects are collected from GitHub. We used two keywords ("pytorch" and "tensorflow") to search projects with more than 200 stars and the programming language set as "Python". Based on the filtering conditions, we initially obtained 1,967 and 1,086 projects related to PyTorch and TensorFlow, respectively. All data were collected up to March 31, 2022. Since testing all the projects is very time-consuming, we performed data cleaning and excluded the following types of projects from our dataset: (1) The project has been archived or deprecated by the developers (PyTorch: 33, TensorFlow: 38). (2) The project does not mention the used version of the DL framework (PyTorch: 812, TensorFlow: 427). (3) The project is based on Keras framework, which provides high-level APIs with a TensorFlow backend (PyTorch: 0, TensorFlow: 222). The projects collected using "tensorflow" may use both TensorFlow and Keras to build their programs. Keras is a high-level DL library, which mainly uses TensorFlow as the backend to implement its functionality. This leads to tight version constraints between Keras and TensorFlow, i.e., changing the Keras version often requires changing the TensorFlow version. (4) The project is difficult to be configured (PyTorch: 1,032, TensorFlow: 349). For example, the link to the dataset provided is no longer available. Following the instructions in README.md, the project still can not be configured for execution. Finally, we have tested 90 PyTorch and 50 TensorFlow projects, as shown in Table I.

TABLE I
Tested DL Projects and Versions of DL Frameworks

DL Framework	#Projects	#Versions	Range	Time Frame
TensorFlow	50	66	0.12.0-2.8.0	up to 03/31/22
PyTorch	90	20	1.0.0-1.11.0	up to 03/31/22

Tested Versions of DL Frameworks. We collected DL framework versions in the Python-PyPI repository, the official distribution site. Similarly, the collected time frame is up to March 31, 2022. As a result, a total of 20 PyTorch versions (1.0.0-1.11.0) and 66 TensorFlow versions (0.12.0-2.8.0) have been collected, as depicted in Table I. Note that all these framework versions can be obtained using the `pip` command, e.g., `pip install torch==1.11.0`.

B. Execution Environment

For each DL project, we deployed the same runtime environment as follows:

- **Host Machine**: The host machine is a workstation with two Intel Xeon Gold 6230R CPUs @ 2.10 GHz (26 cores with 52 threads), 160 GB memory, 256 GB SSD, 8 TB HDD storage, and three Nvidia RTX 2080Ti GPU cards.
- **Host OS**: The OS of the host machine is Ubuntu 18.04.5 LTS Desktop version, a popular Linux distribution.
- **Virtual Environment**: Virtual environments are created using Conda. The version of Conda is 4.4.10.

Then, we configured the dependencies of each project, i.e., Python version, DL framework, other third-party libraries, and the required low-level libraries (e.g., CUDA/cuDNN). To avoid dependency conflicts (e.g., compatibility issues in third-party libraries) impacted by different projects, we used Conda to create a virtual environment for each project and restored the execution environment in it, according to the requirements mentioned in the project.

- **Python Version**: The Python version is determined by the requirements provided in a project. However, many projects in the collected dataset do not provide such information. As aforementioned in the introduction, the Python version will impact the DFVC among DL projects. Thus, to test as many framework versions as

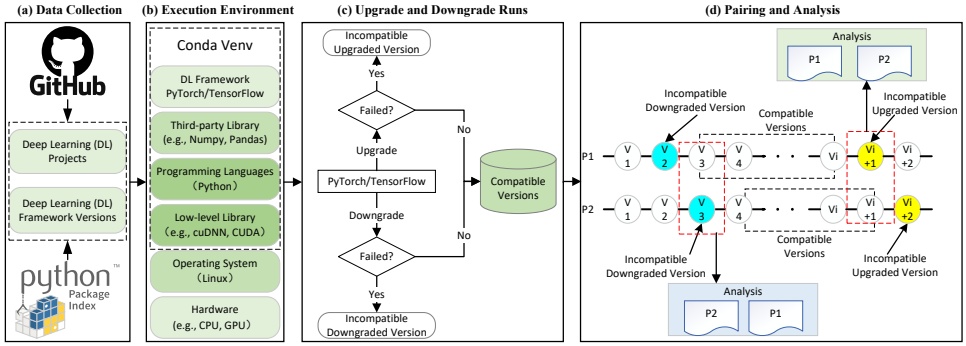


Fig. 3. Overview of our empirical study.

possible, we specified Python-3.6 and Python-3.7 as the Python interpreter environments for the TensorFlow projects and the PyTorch projects, respectively. This is because Python-3.6 is compatible with 60 TensorFlow versions and Python-3.7 is compatible with all the 20 PyTorch versions.

- **DL Framework Version**: The ***starting version*** of DL frameworks is set according to *requirements.txt*, *setup.py*, or *README.md* provided in a project. Note that all the tested projects have starting versions, since projects that do not mention the used DL framework version have been excluded from our dataset.
- **Third-party Library**: To ensure the tested projects can be run normally, we also need to install the third-party libraries that the project depends on. We first checked the third-party libraries and their versions in *requirements.txt*, *setup.py*, or *README.md*. If the project does not provide such information, we relied on the runtime error message, particularly the *ImportError*, to identify the missing dependencies and resolved them until the project can be executed successfully.

C. Upgrade and Downgrade Runs

To test the framework version compatibility of one project, we upgraded and downgraded the framework versions and re-executed the project until it crashed after upgrading or downgrading. In the following, we used an example to describe the details of this part.

We followed the steps described in Sec. III-B to configure the runtime environment for the project cnn-text-classification-tf [39] with a starting version of the DL framework, i.e., **TensorFlow-1.0.0**. Besides, to avoid potential dependency conflicts between the upgrading and downgrading environments, we used Conda to generate two virtual environments for upgrading and downgrading, respectively.

For the upgrade runs, we first updated the starting version (1.0.0) to the next version (1.0.1) in the virtual environment using the `pip` command (e.g., pip install tensorflow==1.0.1). Then, we recorded the third-party libraries and their versions provided by `conda list` command in the virtual environment. Note that we only upgraded the DL framework version. Since the DL framework also has dependencies with other third-party libraries, upgrading the framework version could also change the versions of some third-party libraries. After upgrading the framework version, we continued to run the project and recorded the screen print information. Repeating the above steps until the project fails to run properly in two consecutive upgrades, we stopped the experiments. For example, cnn-text-classification-tf works fine in versions ranging from 1.0.0 to 1.4.1 but crashes in versions 1.5.0 and 1.5.1. Similarly, for the downgrade runs, the procedure is the same as the abovementioned descriptions for the upgrade runs. For cnn-text-classification-tf, it works fine in the starting version, i.e., 1.0.0, but fails in versions 0.12.0 and 0.12.1.

It should be noted that the upgrading and downgrading experiments will be terminated after the project cannot be executed in both the two consecutive framework versions. This is because the installation package of PyTorch-1.8.0 in the Python-PyPI repository lacks the CUDA/cuDNN runtime libraries for the Nvidia GPU architecture `sm-75` (e.g., the Nvidia RTX 2080Ti cards in our experiment environment) [40]. As a result, all the PyTorch projects cannot be executed normally in version 1.8.0. Thus, to mitigate this impact, the experiment would be terminated if the project cannot normally run in two consecutive framework versions after upgrading/downgrading. Another thing worth noting is that a project does not work properly could also due to the incompatibility between the Python version specified in the project and the corresponding upgraded or downgraded framework versions.

Besides, due to factors such as large training datasets or complex neural networks with multiple layers, executing these projects can be time-consuming. For instance, testing a single framework version may require several hours of computation. To accelerate the experiments, we made adjustments to the projects by reducing the number of training epochs or iterations (e.g., modifying epoch settings) or dividing the training dataset into smaller subsets. For example, for the cnn-text-classification-tf project, we modified the default epoch value of 200 to 1.

Moreover, since DL projects often consist of multiple source files and invoke various framework APIs, the compatible framework versions determined by running a single command may not be representative of the overall framework compatibility of the entire project. Therefore, to cover more framework APIs, we executed all the commands

mentioned in README.md that are related to DL training, testing, and evaluation. For example, for the cnn-text-classification-tf project, we executed the two commands listed in README.md for training and testing.

After finishing the above steps, we recorded the framework-specific version information for the upgrade and downgrade process, as illustrated in Fig. 3(d):

- **Incompatible Upgraded Version (IUV)**: We refer to the initial framework version among two consecutive updates, where the project did not execute properly, as the incompatible upgraded version (IUV). The project is unable to execute successfully under both the IUV and the subsequent newer version during the upgrade process. However, it functions properly when operating under an older version that is prior to the IUV. For instance, TensorFlow-1.5.0 is identified as the IUV of cnn-text-classification-tf.
- **Incompatible Downgraded Version (IDV)**: Likewise, we refer to the initial framework version among two consecutive downgrades, where the project did not operate as intended, as the incompatible downgraded version (IDV). During the downgrade process, the project did not work properly under both the IDV and the older released version. In contrast, it performs well with a newer version that is more recent than the IDV. For example, TensorFlow-0.12.1 is the IDV of cnn-text-classification-tf.
- **Compatible Versions**: The framework versions in which projects can run smoothly are considered compatible versions. For instance, cnn-text-classification-tf is compatible with a range of TensorFlow versions from 1.0.0 to 1.4.1, encompassing a total of 8 versions.

D. Pairing and Analysis

Identification of DFVC Pairs. To investigate the root causes of DFVC, we first identified all the project pairs, where one project can run successfully with a specific version while the other project fails to do so. The pairing process is described as follows. For the example shown in Fig. 3(d), given two projects $P1$ and $P2$, the IUV of $P1$ belongs to the compatible versions of $P2$, and the IDV of $P2$ is compatible with $P1$. Thus we obtain two pairs $(P1, P2)$ and $(P2, P1)$, respectively. Note that for the pair (x, y), x represents the project is incompatible with a specific framework version, while y denotes the project is compatible with that version.

We finally obtained 6,926 pairs and further classified them into three categories (Table II): (1) project y and project x use different Python versions, (2) project y does not call the framework API involved in project x's error message, and (3) project y calls the framework API involved in project x's error message. The labeling details are presented as follows:

- **Step 1**: For a pair (x, y), if the Python version of project x is incompatible with its IUV or IDV, the pair will be classified as category (1); otherwise, we investigated the traceback message of project x to determine the framework API that causes project x to crash. For example, Fig. 4 shows the traceback message of cnn-text-classification-tf. We can see that the TensorFlow API `concat()` is called in the client code, which is the closest to the *TypeError* message. Thus, the framework API that caused the project to crash is `concat()`.
- **Step 2**: After obtaining the framework API in step 1, we wrote a script to match the API in the source code of project y automatically to determine whether the API is also used by project x. If the API does not match, the pair is classified as category (2); otherwise, we proceed with the following step for further analysis.
- **Step 3**: Although project y uses the framework API that caused project x to crash, it does not necessarily imply that the API is invoked during runtime. The upgrade and downgrade runs may not cover the API invocations. Therefore, we manually inserted a print statement, i.e., print("compatibility"), before the API calling statement in project y. We then executed project y to determine whether the inserted token was printed during runtime. If the token was printed, the pair will be labeled as category (3); otherwise, it is classified as category (2).

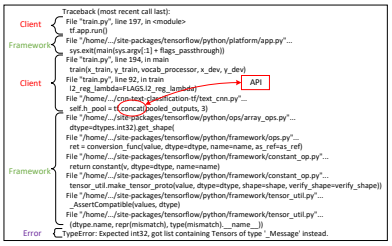


Fig. 4. Traceback of cnn-text-classification-tf executed in TensorFlow-0.12.1.

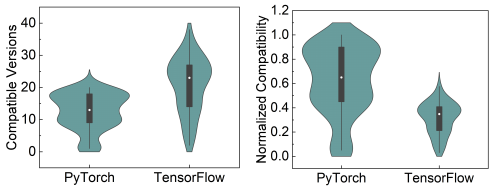
TABLE II
DISTRIBUTION OF DFVC PAIRS

Category	(1) Python	(2) w/o. the same API	(3) w. the same API
#Pairs	1,415	4,627	884
%Percentage	20.4%	66.8%	12.8%

Investigation of Root Causes. Since the reasons behind the types (1) and (2) DFVC pairs are relatively apparent (i.e., the Python version and the absence of using the same breaking API), our focus shifted towards analyzing the root causes of type (3) pairs.

First, we divided 884 pairs of category (3) into 54 groups based on the unique x in pair (x, y), i.e., 54 projects. For each group, we analyzed the cause of the compatibility issue of the project x. Then, we compared the client code of project y that calls the same framework API, as well as the third-party libraries and their versions, as described in Sec. III-C, to investigate why projects x and y have DFVC, i.e., project x crashes but project y works normally under the same framework version.

Then, by modifying the client code of project y or the related third-party libraries and versions, we determined whether project y also has the same compatibility issue with the framework version that project x does not execute properly. For cross-validation, we also modified the calling framework API in the client code or third-party libraries and versions of project x according to why project y can run properly (e.g., API import path and the installed third-party library). This can



(a) Within-Framework (b) Cross-Frameworks

Fig. 5. Distribution of compatible versions of PyTorch and TensorFlow projects.

further verify whether project x can run properly and validate the identified root cause of the DFVC pair.

IV. Results and Analysis

A. RQ1: Prevalence of Difference in Framework Version Compatibility Among DL Projects

To analyze the prevalence of DFVC, we compared the framework version compatibility from two aspects: compatibility differences within the same framework and cross frameworks, as shown in Fig. 5.

Compatibility Differences Within the Same DL Framework. Fig. 5(a) shows the distribution of the number of compatible framework versions of PyTorch and TensorFlow projects, respectively. It can be seen that both PyTorch and TensorFlow projects differ in the number of framework versions that they are compatible with. The result is expected, as mentioned in the introduction section, the differences in the framework API used by each project and the version of Python used for project development can lead to DFVC.

Compatibility Differences Cross DL Frameworks. Since the number (66) of TensorFlow versions tested is larger than the number (20) of PyTorch versions, the number of compatible framework versions for the TensorFlow projects shown in Fig. 5(a) is larger than those of the PyTorch projects. However, this can not imply that the framework version compatibility of TensorFlow is better than that of PyTorch. Therefore, we normalized the number of compatible framework versions of projects, i.e., the number of compatible versions/total number of framework versions, to evaluate the compatibility of TensorFlow and PyTorch framework versions. As depicted in Fig. 5(b), the PyTorch projects have better framework version compatibility than the TensorFlow projects.

> **Answer to RQ1**: The difference in framework version compatibility is prevalent among DL projects. The number of compatible framework versions for the tested PyTorch projects ranges from 1 to 20, while the number for the tested TensorFlow projects ranges from 2 to 38. The framework version compatibility of PyTorch projects is better than that of TensorFlow projects.

B. RQ2: Root Causes of Difference in Framework Version Compatibility Among DL Projects

We summarized the root causes of DFVC into seven types: including Python version, absence of using the same breaking API, import path, parameter, third-party library, resource, and API usage constraint, as depicted in Fig. 6. Note that the five root causes import path, parameter, third-party library, resource, and API usage constraint are identified through analyzing all the category (3) DFVC pairs, as described in Sec. III-D. In the following, we will discuss each type in detail.

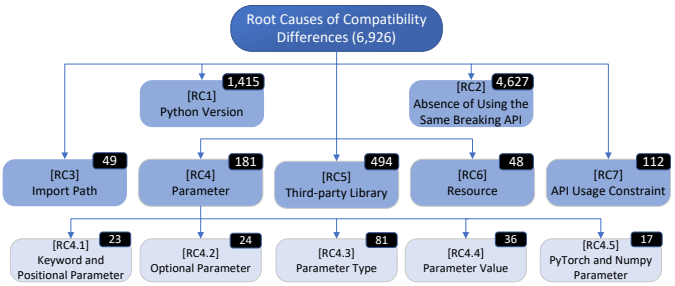
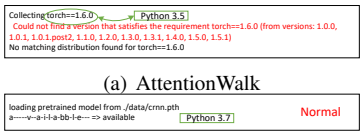


Fig. 6. Root causes of compatibility differences.



(a) AttentionWalk

(b) crnn.pytorch

Fig. 7. Execution results of AttentionWalk [41] and crnn.pytorch [42] in PyTorch-1.6.0.

1) Root Cause 1: Python Version: **Definition: the difference in the framework version compatibility for DL projects is related to the Python version.** As described in the Introduction, DL projects may have DFVC if they were developed using different Python versions. Fig. 7 shows another example. The PyTorch project AttentionWalk [41] was created by Python-3.5. However, PyTorch-1.6.0's installation package is not available in the PyPI repository for Python-3.5. Thus, attempting to install PyTorch-1.6.0 with the `pip` command in a Python-3.5 environment will lead to the failure of PyTorch installation: *ERROR: No matching distribution found for torch==1.6.0.* In contrast to AttentionWalk, project crnn.pytorch [42] was developed using Python-3.7, which is compatible with PyTorch-1.6.0. As a result, the project can execute successfully in PyTorch-1.6.

2) Root Cause 2: Absence of Using the Same Breaking API: **Definition: the difference in the framework version compatibility for DL projects is related to the absence of using the same breaking API.** If two DL projects utilized different APIs, it is intuitive that they are likely to have DFVC. For instance, Group-Normalization-Tensorflow [43] and cnn_captcha [17], two projects based on TensorFlow-1.3.0 and TensorFlow-1.7.0, respectively. The project Group-Normalization-Tensorflow crashes at runtime when downgrading the TensorFlow version to 1.0.1, but cnn_captcha continues to function as intended. By analyzing the traceback message, it is found that Group-Normalization-Tensorflow invokes the optimizing loss function `optimize_loss()` with parameter `increment_global_step`, which was introduced in TensorFlow version 1.1 [44]. Therefore, Group-Normalization-Tensorflow cannot work properly in a TensorFlow-1.0.1 environment. Conversely, cnn_captcha runs normally under TensorFlow-1.0.1, since it does not call the `optimize_loss()` function.

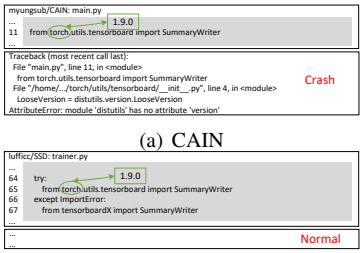


Fig. 8. Execution results of CAIN [45] and SSD [46] in PyTorch-1.9.0.

3) Root Cause 3: Import Path: **Definition: the difference in the framework version compatibility for DL projects is related to the API import path.** APIs in a DL framework could be added, removed, or relocated with the framework's evolution. Therefore, importing APIs from different paths may lead to DFVC, as the example illustrated in Fig. 1.

Fig. 8 shows another case where the project CAIN [45] crashes in PyTorch-1.9.0 because it uses the following line of code: *from torch.utils.tensorboard import SummaryWriter*. However, the project SSD [46] also uses this code but runs properly. By comparing the client-side code, it is found that SSD implements an exception catch statement to catch the error raised by *from torch.utils. tensorboard import SummaryWriter* and replaces it with *from tensorboardX import SummaryWriter* to avoid the error. CAIN is incompatible with PyTorch-1.9.0 because the module Distutils is no longer imported from Python since PyTorch-1.9.0, but instead, it is imported from Setuptools [47]. However, the implementation of the Distutils in Setuptools is different from the one in the Python standard library. For example, the Distutils module in Setuptools does not contain the `version` attribute. Consequently, *AttributeError* will occur (Fig. 8(a)).

4) Root Cause 4: Parameter: **Definition: the difference in the framework version compatibility for DL projects is related to API parameters.** API Parameters are frequently changing during the evolution of DL frameworks, leading to parameter-related compatibility issues that are prevalent in DL projects [13]. In particular, we observed the following five subtypes of root cause 4:

- **Keyword and Positional Parameter.** *Definition: the difference in the framework version compatibility for DL projects is related to the use of keyword and positional parameters.* In Python programs, APIs can be called with both keyword and positional parameters. As the framework evolves, compatibility issues may appear if keyword parameters have been added, renamed, or removed. Therefore, using keywords or parameter positions for parameter passing may lead to DFVC. For the examples shown in Fig. 2 and Fig. 9, SimpleHTR [15] crashes when calling `conv2d()` in TensorFlow-1.13.2, while projects cnn_captcha [17] and neural-style [48] works normally. The commonality of the two projects is that they all use the positional parameter to pass the output dimension of the convolutional layer. Still, SimpleHTR uses the keyword parameter (i.e., `filters`), which is added as an alias of the keyword parameter `filter` in TensorFlow-1.14.0 [20]. Therefore, SimpleHTR is incompatible with TensorFlow-1.13.2, while projects cnn_captcha and neural-style are compatible with the framework version.

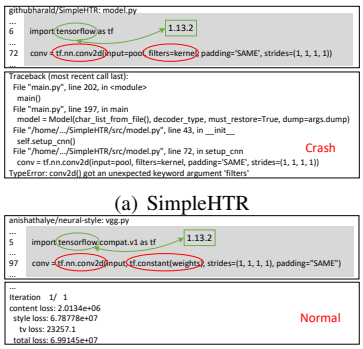


Fig. 9. Execution results of SimpleHTR [15] and neural-style [48] in TensorFlow-1.13.2.

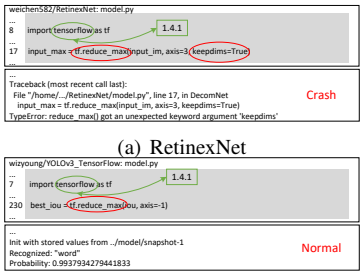
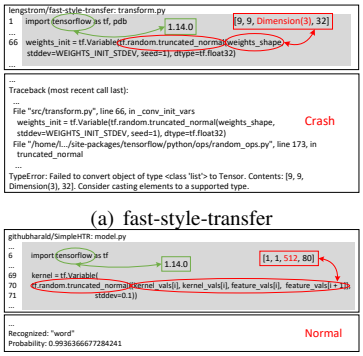


Fig. 10. Execution results of RetinexNet [16] and YOLOv3_TensorFlow [49] in TensorFlow-1.4.1.

- **Optional Parameter.** *Definition: the difference in the framework version compatibility for DL projects is related to whether the optional parameter is used.* Some parameters of DL framework APIs often have default values, i.e., optional parameters, which can be used without passing values when invoking the APIs. Along with the framework's evolution, optional parameters of APIs may be removed or renamed. Therefore, employing keyword-based value passing for optional parameters may lead to DFVC. As depicted in Fig. 10, the project RetinexNet [16] calls `reduce_max()` in TensorFlow-1.4.1 causing the project to crash. `reduce_max()` is used to compute the maximum of elements across the dimensions of a tensor. However, the project YOLOv3_TensorFlow [49] also calls `reduce_max()` under TensorFlow-1.4.1, but it works normally. The reason is that the two projects use the optional parameter differently. RetinexNet uses the keyword parameter `keepdims`, while YOLOv3_TensorFlow utilizes the default value (`None`) for this optional parameter. In the evolution of TensorFlow, the keyword `keepdim` was renamed to `keepdims` in 1.5.0 [18]. Therefore, using the keyword `keepdims` before version 1.5.0 (e.g., 1.4.1) will result in the following error: *TypeError: reduce_max() got an unexpected keyword argument 'keepdims'*, as shown in Fig. 10.

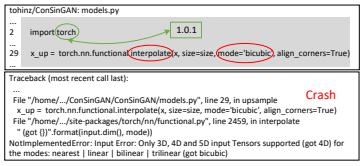


(a) fast-style-transfer

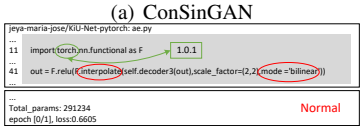
(b) SimpleHTR

Fig. 11. Execution results of fast-style-transfer [50] and SimpleHTR [15] in TensorFlow-1.14.0.

- **Parameter Type.** *Definition: the difference in the framework version compatibility for DL projects is related to parameter types.* When calling the API, different value types passing to parameters could lead to DFVC. Fig. 11 shows that the project fast-style-transfer [50] crashes in TensorFlow-1.14.0 when calling `truncated_normal()`, a function used to generate truncated normally distributed random numbers. However, the project SimpleHTR [15] also calls `truncated_normal()` but is compatible with the same framework version. Further investigation reveals that the type of input value assigned to the parameter `shape` is different. In the case of fast-style-transfer, the input for the `shape` parameter is [9, 9, Dimension(3), 32], where the third index corresponds to the type *tensorflow.python.framework.Dimension*. On the other hand, SimpleHTR utilizes [1, 1, 512, 80] as the input for the `shape` parameter, with the third index being type *int*. The *dimension* type is not supported in TensorFlow-1.14.0, which was fixed in 1.15.0 [51].
- **Parameter Value.** *Definition: the difference in the framework version compatibility for DL projects is related to parameter values.* As depicted in Fig. 12, the projects ConSinGAN [52] and KiU-Net-pytorch [21] exhibit different framework version compatibility with the tensor interpolation operation `interpolate()` when using PyTorch-1.0.1. This is because the two projects invoke `interpolate()` with distinct input values for the `mode` parameter. Specifically, ConSinGAN inputs `bicubic`, while KiU-Net-pytorch inputs `bilinear`. Note that the `bicubic` feature was introduced in PyTorch-1.1.0 [53]. As a result, employing `bicubic` as the value for the `mode` parameter before version 1.1.0 leads to the *NotImplementedError* in Fig. 12(a).
- **PyTorch and Numpy Parameter.** *Definition: the difference in the framework version compatibility for DL projects is related to using the PyTorch and Numpy parameters.* DL projects often involve a large number of tensor operations. Both PyTorch and Numpy define many functions for tensor operations, such as `sum()`, `max()`, etc. Parameter names between the tensor operations functions defined in PyTorch and Numpy are slightly

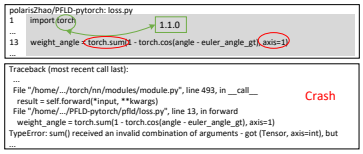


(a) ConSinGAN

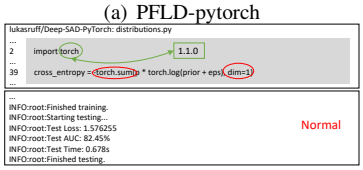


(b) KiU-Net-pytorch

Fig. 12. Execution results of ConSinGAN [52] and KiU-Net-pytorch [21] in PyTorch-1.0.1.



(a) PFLD-pytorch

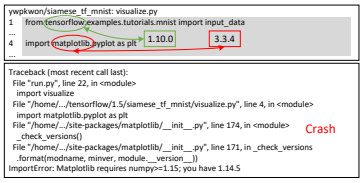


(b) Deep-SAD-PyTorch

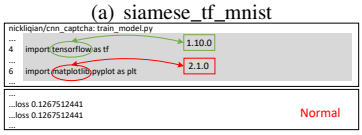
Fig. 13. Execution results of PFLD-pytorch [56] and Deep-SAD-PyTorch [57] in PyTorch-1.1.0.

different. To facilitate user convenience [54], automatic translations of parameter names in Numpy to the corresponding PyTorch APIs were introduced in PyTorch-1.2.0 [55]. For example, the parameter `axis` in Numpy is automatically translated to `dim` in PyTorch APIs. Therefore, if one project uses the Numpy parameter in PyTorch APIs with a version before 1.2.0, it will encounter a crash. Conversely, if the project uses PyTorch parameters, it will work normally, as shown in Fig. 13.

5) Root Cause 5: Third-party Library: **Definition: the difference in the framework version compatibility for DL projects is related to third-party libraries and their versions.** Due to the dependency constraints of DL frameworks on third-party libraries, the version of these third-party libraries could be changed when upgrading or downgrading the framework versions. This may lead to version conflicts between these libraries and other third-party libraries. Fig. 14 shows that the project siamese_tf_mnist [58] fails to import Matplotlib in TensorFlow-1.10.0, while cnn_captcha [17] imports Matplotlib normally with the same framework version. By comparing the record of `conda list` in the starting version's virtual environment, we found that the installed Matplotlib versions for siamese_tf_mnist and cnn_captcha are 3.3.4 and 2.1.0, respectively. Note that TensorFlow-1.10.0 requires the compatible Numpy version satisfying (>=1.13.3,<=1.14.5). When TensorFlow-1.10.0 is installed, it will change the Numpy version to 1.14.5. However, the compatible Numpy versions for Matplotlib versions 2.1.0 and 3.3.4 are (>= 1.7.1) and (>=1.15), respectively. Consequently, the dependency conflict between

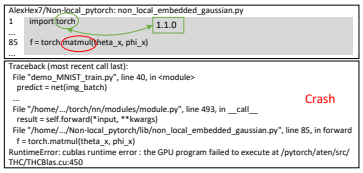


(a) siamese_tf_mnist

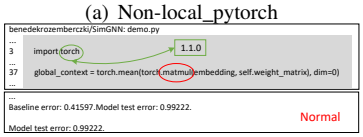


(b) cnn_captcha

Fig. 14. Execution results of siamese_tf_mnist [58] and cnn_captcha [17] in TensorFlow-1.10.0.



(a) Non-local_pytorch



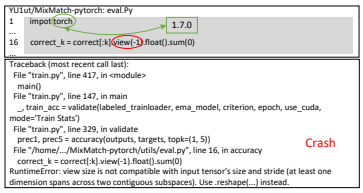
(b) SimGNN

Fig. 15. Execution results of Non-local_pytorch [59] and SimGNN [60] in PyTorch-1.1.0.

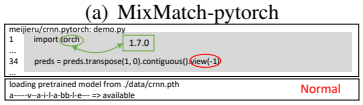
Numpy and Matplotlib occurs in the siamese_tf_mnist project.

6) Root Cause 6: Resource: **Definition: the difference in the framework version compatibility for DL projects is related to computing resources.** During the evolution of DL frameworks, there is a continuous effort to optimize APIs, which often leads to reduced computing resources required for completing operations. Therefore, when two DL projects require different computing resources, they may experience DFVC after downgrading the framework version. Fig. 15 shows that the project Non-local_pytorch [59] calls `matmul()` (matrix product of two tensors) with an error in PyTorch-1.1.0, while SimGNN [60] runs normally. Further investigation revealed that the Non-local_pytorch project requires more computing resources than SimGNN when performing the matrix product operation. PyTorch optimized the implementation code of `matmul()` in version 1.2.0 [61], which makes the matrix product operation of tensors with improved performance running on both CPU and GPU platforms [61]. Therefore, when the PyTorch version was downgraded from 1.2.0 to 1.1.0, Non-local_pytorch would encounter the *RuntimeError*, as depicted in Fig. 15(a).

7) Root Cause 7: API Usage Constraint: **Definition: the difference in the framework version compatibility for DL projects is related to API usage constraints.** Several APIs in DL frameworks have specific requirements for the data to be processed, such as `view()` in PyTorch for transforming the shape of a tensor. The API requires that the processed tensor is stored contiguously in memory; otherwise, a runtime error occurs [62]. As shown in Fig. 16, both MixMatch-pytorch [63] and crnn.pytorch [42] invoke `view()` in PyTorch-1.7.0, but



(a) MixMatch-pytorch



(b) crnn.pytorch

Fig. 16. Execution results of MixMatch-pytorch [63] and crnn.pytorch [42] in PyTorch-1.7.0.

the latter project works properly. This is because MixMatch-pytorch calls `view(-1)` with a non-continuous tensor, whereas crnn.pytorch employs the `contiguous()` function to convert the tensor into a continuous one before invoking `view(-1)`.

> **Answer to RQ2**: The root causes of the difference in framework version compatibility among DL projects include Python version, absence of using the same breaking API, import path, parameter, third-party library, resource, and API usage constraint.

V. Implications

Based on our empirical analysis, we summarized six implications for DL practitioners.

- **Implication #1**: PyTorch exhibits better framework version compatibility compared to TensorFlow. For achieving better framework version compatibility of DL projects, PyTorch can be chosen as the development framework. Besides, it is recommended to choose a specific Python version, which is compatible with more framework versions. For example, Python-3.6 for TensorFlow projects (compatible with most versions from 0.12.1 to 2.6.2), while Python-3.7 for PyTorch projects (compatible with versions from 1.0.0 to 1.11.0). This is because these versions demonstrate better compatibility with their respective frameworks.
- **Implication #2**: Developers should be mindful of potential changes in API import paths when switching between different framework versions. It is important to update the API import paths accordingly to ensure compatibility and avoid runtime errors. For example, to improve the framework version compatibility of TensorFlow-1.X projects against TensorFlow-2.X versions, the APIs should be imported from `tensorflow.compat.v1`.
- **Implication #3**: To enhance the forward and backward compatibility of DL framework versions, developers are suggested to utilize a try and except statement to catch exceptions when encountering compatibility issues after upgrading or downgrading the framework version. Rather than directly modifying the problematic code, this

approach allows for fault tolerance, enabling the project to handle compatibility issues caused by framework version changes gracefully.

- **Implication #4**: When upgrading or downgrading a DL framework, it is crucial to carefully examine whether the APIs used comply with the requirements of the new framework version. In particular, developers should pay attention to the parameters, including keyword and positional parameters, parameter types, and parameter values. It is essential to ensure that the API calls align with the specifications and changes introduced in the updated or downgraded framework version.
- **Implication #5**: When upgrading or downgrading DL framework versions, developers should pay attention to the changes in associated third-party libraries. To avoid dependency conflicts between third-party libraries, developers can leverage various tools, such as PyEGo [64], smartPip [65], and Watchman [66], which can assist in automatically managing and resolving version conflicts between different third-party libraries.
- **Implication #6**: To improve the framework version compatibility of DL projects, developers should be mindful of API usage constraints or use static value-flow analysis [67] to reason about the constraints. For example, when using the `view()` operation to manipulate a tensor in PyTorch projects, it is suggested to invoke `contiguous()` function beforehand to ensure that the data is stored contiguously in memory. By adhering to recommended usage patterns and considering the constraints imposed by specific APIs, developers can minimize compatibility issues and ensure smoother transitions between framework versions.

Note that, since the DL projects examined in this paper are all based on Python development, some findings and implications might also hold relevance for non-DL Python projects. Particularly, implications #1, #2, and #6 are specifically proposed for DL projects.

VI. Threats to Validity

Internal Threat. The threat to internal validity mainly comes from our experiments. First, the tested compatible framework versions were obtained by executing the commands provided in the projects. However, the used API in the project may not be covered by the testing. To reduce this threat, we performed all the commands of the project. Besides, the determination of the error-induced API for DFVC pairing is considered another threat. For example, for some DL projects, it is difficult to trace the error-induced API, such as a variable assigned by an API invoking. This could impact the identification of DFVC pairs that used the same API. Moreover, Our methodology focuses on investigating the DFVC induced by software-related issues rather than hardware-related issues.

External Threat. The threat to external validity mainly comes from the generalization of our finding results. To reduce this threat, we tested 140 DL projects based on the two most representative DL frameworks (i.e., TensorFlow and PyTorch) with all officially released framework versions in the Python-PyPI repository. However, the choice of only two frameworks may limit the generalization of our finding results. In the future, we plan to investigate more projects and frameworks to enhance our empirical results.

VII. Related Work

Python is a popular programming language that offers a wide range of third-party libraries. Over time, the APIs of third-party libraries may undergo changes, which can result in compatibility issues when upgrading/downgrading library versions in client code. Existing research has primarily focused on analyzing API evolution trends and detecting breaking changes [13], [23], [24], detecting and fixing deprecated APIs [25]–[27], [68], [69]. Zhang *et al.* [13] analyzed the API evolution patterns by extracting APIs from six typical Python frameworks (i.e., TensorFlow, Keras, Scikit-Learn, Pandas, Flask, and Django) and found that breaking changes caused by Python API evolution are distinct from Java APIs. Haryono *et al.* [26] characterized the deprecated APIs by conducting an empirical study of 112 deprecated APIs from three popular machine learning libraries (i.e., Scikit-Learn, TensorFlow, and PyTorch). They found that the deprecated APIs are updated mainly from three dimensions: operation, API mapping, and context dependency. They further proposed MLCatchUp [27] to can automate the update of deprecated API usages by inferring the transformation between deprecated and updated API signatures. Our study differs from the aforementioned literature in that it is the first empirical study of the difference in framework version compatibility among DL projects.

VIII. Conclusion

In this paper, we conducted the first empirical study to investigate the difference in framework version compatibility among DL projects. We tested 90 PyTorch and 50 TensorFlow projects with 20 PyTorch and 66 TensorFlow framework versions. By analyzing the experiment results, we summarized seven root causes, i.e., Python version, absence of using the same breaking API, import path, parameter, third-party library, resource, and API usage constraint, to answer why DL projects differ in compatible framework versions. We further provided six implications for DL practitioners. We believe this study can provide a better understanding of the difference in framework version compatibility for DL projects.

Acknowledgment

We thank the anonymous reviewers for their valuable comments. This work was supported in part by National Natural Science Foundation of China under Grant 62002163, Natural Science Foundation of Jiangsu Province under Grant BK20200441, Australian Research Council under Grant DP210101348, and National Key Research and Development Program of China under Grant 2019YFE0198100. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the above sponsoring entities.

REFERENCES

[1] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 8599–8603.

[2] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, 2016.

[3] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[4] Y. Sui, X. Cheng, G. Zhang, and H. Wang, "Flow2vec: Value-flow-based precise code embedding," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–27, 2020.

[5] C. Watson, N. Cooper, D. N. Palacio, K. Moran, and D. Poshyvanyk, "A systematic literature review on the use of deep learning in software engineering research," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 2, pp. 1–58, 2022.

[6] G. Xiao, X. Du, Y. Sui, and T. Yue, "Hindbr: Heterogeneous information network based duplicate bug report prediction," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 195–206.

[7] X. Du, Z. Zheng, G. Xiao, Z. Zhou, and K. S. Trivedi, "Deepsim: Deep semantic information-based automatic mandelbug classification," *IEEE Transactions on Reliability*, vol. 71, no. 4, pp. 1540–1554, 2021.

[8] J. Zhu, G. Xiao, Z. Zheng, and Y. Sui, "Enhancing traceability link recovery with unlabeled data," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, 2022, pp. 446–457.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "{TensorFlow}: a system for {Large-Scale} machine learning," in *Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–283.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[11] pypi.org, "Pypi," Retrieved May 10, 2023 from https://pypi.org/, 2023.

[12] Y. Deng, C. Yang, A. Wei, and L. Zhang, "Fuzzing deep-learning libraries via automated relational api inference," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2022, pp. 44–56.

[13] Z. Zhang, H. Zhu, M. Wen, Y. Tao, Y. Liu, and Y. Xiong, "How do python framework apis evolve? an exploratory study," in *Proceedings of 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 81–92.

[14] GitHub.com, "hwalsuklee/tensorflow-fast-style-transfer," Retrieve May 10, 2023 from https://github.com/hwalsuklee/tensorflow-fast-style-transfer, 2023.

[15] ——, "githubharald/simplehtr," Retrieve May 10, 2023 from https://github.com/githubharald/SimpleHTR, 2023.

[16] ——, "weichen582/retinexnet," Retrieve May 10, 2023 from https://github.com/weichen582/RetinexNet, 2023.

[17] ——, "nickliqian/cnn_captcha," Retrieve May 10, 2023 from https://github.com/nickliqian/cnn_captcha, 2023.

[18] ——, "commit b1d8c59 of tensorflow," Retrieved May 10, 2023 from https://github.com/tensorflow/tensorflow/commit/b1d8c59e9b014b527fb2fbef9ce9afc14dbc4938, 2023.

[19] ——, "Tensorflow 2.0.0," Retrieved May 10, 2023 from https://github.com/tensorflow/tensorflow/releases/tag/v2.0.0, 2023.

[20] ——, "commit 13679b8 of tensorflow," Retrieve May 10, 2023 from https://github.com/tensorflow/tensorflow/commit/13679b875aa87a3d28cf224388ee6a255d3f8844, 2023.

[21] ——, "jeya-maria-jose/kiu-net-pytorch," Retrieve May 10, 2023 from https://github.com/jeya-maria-jose/KiU-Net-pytorch, 2023.

[22] ——, "hlwang1124/sne-roadseg," Retrieve May 10, 2023 from https://github.com/hlwang1124/SNE-RoadSeg, 2023.

[23] Z. Zhang, Y. Yang, X. Xia, D. Lo, X. Ren, and J. Grundy, "Unveiling the mystery of api evolution in deep learning frameworks: a case study of tensorflow 2," in *Proceedings of 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021, pp. 238–247.

[24] X. Du and J. Ma, "Aexpy: Detecting api breaking changes in python packages," in *Proceedings of 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, 2022, pp. 470–481.

[25] J. Wang, L. Li, K. Liu, and H. Cai, "Exploring how deprecated python library apis are (not) handled," in *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020, pp. 233–244.

[26] S. A. Haryono, F. Thung, D. Lo, J. Lawall, and L. Jiang, "Characterization and automatic updates of deprecated machine-learning api usages," in *Proceedings of 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 137–147.

[27] ——, "Mlcatchup: Automated update of deprecated machine-learning apis in python," in *Proceedings of 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 584–588.

[28] slant.co, "which-is-the-best-os-for-deep-learning," Retrieve May 10, 2023 from https://www.slant.co/topics/9702/~which-is-the-best-os-for-deep-learning, 2023.

[29] python.org, "python," Retrieved May 10, 2023 from https://www.python.org/, 2023.

[30] X. Du, G. Xiao, and Y. Sui, "Fault triggers in the tensorflow framework: An experience report," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 1–12.

[31] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with numpy," *Nature*, 2020.

[32] W. McKinney *et al.*, "Pandas: A foundational python library for data analysis and statistics," *PyHPC*, 2011.

[33] G. Bradski, "The opencv library." *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[34] python pillow.org, "Pillow," Retrieve May 10, 2023 from https://python-pillow.org, 2023.

[35] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.

[36] seaborn.pydata.org, "Seaborn," Retrieve May 10, 2023 from https://seaborn.pydata.org, 2023.

[37] G. Xiao, J. Liu, Z. Zheng, and Y. Sui, "Nondeterministic impact of cpu multithreading on training deep learning systems." in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, 2021, pp. 557–568.

[38] L. Yin, Y. Zhang, Z. Zhang, Y. Peng, and P. Zhao, "Parax: Boosting deep learning for big data analytics on many-core cpus," *Proceedings of the VLDB Endowment*, pp. 864–877, 2021.

[39] GitHub.com, "dennybritz/cnn-text-classification-tf," Retrieve May 10, 2023 from https://github.com/dennybritz/cnn-text-classification-tf, 2023.

[40] discuss.pytorch.org, "runtimeerror-cudnn-error-cudnn-status-not-initialized," Retrieve May 10, 2023 from https://discuss.pytorch.org/t/runtimeerror-cudnn-error-cudnn-status-not-initialized, 2023.

[41] GitHub.com, "benedekrozemberczki/attentionwalk," Retrieve May 10, 2023 from https://github.com/benedekrozemberczki/AttentionWalk, 2023.

[42] ——, "meijieru/crnn.pytorch," Retrieve May 10, 2023 from https://github.com/meijieru/crnn.pytorch, 2023.

[43] ——, "shaohua0116/group-normalization-tensorflow," Retrieve May 10, 2023 from https://github.com/shaohua0116/Group-Normalization-Tensorflow, 2023.

[44] ——, "commit 35a4183 of tensorflow," Retrieve May 10, 2023 from https://github.com/tensorflow/tensorflow/commit/35a4183cf261a3893549c7877f63fc415e8287ed, 2023.

[45] ——, "myungsub/cain," Retrieve May 10, 2023 from https://github.com/myungsub/CAIN, 2023.

[46] ——, "lufficc/ssd," Retrieve May 10, 2023 from https://github.com/lufficc/SSD, 2023.

[47] setuptools.pypa.io, "Setuptools," Retrieved May 10, 2023 from https://setuptools.pypa.io/, 2023.

[48] GitHub.com, "anishathalye/neural-style," Retrieve May 10, 2023 from https://github.com/anishathalye/neural-style, 2023.

[49] ——, "wizyoung/yolov3_tensorflow," Retrieve May 10, 2023 from https://github.com/wizyoung/YOLOv3_TensorFlow, 2023.

[50] ——, "lengstrom/fast-style-transfer," Retrieve May 10, 2023 from https://github.com/lengstrom/fast-style-transfer, 2023.

[51] ——, "commit c4f40ae of tensorflow," Retrieved May 10, 2023 from https://github.com/tensorflow/tensorflow/commit/c4f40aea1d4f916aa3dfeb79f024c495ac609106, 2023.
[52] ——, "tohinz/consingan," Retrieve May 10, 2023 from https://github.com/tohinz/ConSinGAN, 2023.
[53] ——, "commit 59d71b9 of pytorch," Retrieved May 10, 2023 from https://github.com/pytorch/pytorch/commit/59d71b9664b57b0ea0de0d87cea87b21daa4dd7b, 2023.
[54] ——, "pull 20451 of pytorch," Retrieved May 10, 2023 from https://github.com/pytorch/pytorch/pull/20451, 2023.
[55] ——, "commit 51eba78 of pytorch," Retrieved May 10, 2023 from https://github.com/pytorch/pytorch/commit/51eba7824cc39ac58859d81de4ac3fdf10ae7fae, 2023.
[56] ——, "polariszhao/pfld-pytorch," Retrieve May 10, 2023 from https://github.com/polarisZhao/PFLD-pytorch, 2023.
[57] ——, "lukasruff/deep-sad-pytorch," Retrieve May 10, 2023 from https://github.com/lukasruff/Deep-SAD-PyTorch, 2023.
[58] ——, "ywpkwon/siamese_tf_mnist," Retrieve May 10, 2023 from https://github.com/ywpkwon/siamese_tf_mnist, 2023.
[59] ——, "Alexhex7/non-local_pytorch," Retrieve May 10, 2023 from https://github.com/AlexHex7/Non-local_pytorch, 2023.
[60] ——, "benedekrozemberczki/simgnn," Retrieve May 10, 2023 from https://github.com/benedekrozemberczki/SimGNN, 2023.
[61] ——, "Pytorch 1.2.0," Retrieved May 10, 2023 from https://github.com/pytorch/pytorch/releases/tag/v1.2.0, 2023.
[62] ——, "Pytorch 1.6.0," Retrieved May 10, 2023 from https://github.com/pytorch/pytorch/releases/tag/v1.6.0, 2023.
[63] ——, "Yu1ut/mixmatch-pytorch," Retrieve May 10, 2023 from https://github.com/YU1ut/MixMatch-pytorch, 2023.
[64] H. Ye, W. Chen, W. Dou, G. Wu, and J. Wei, "Knowledge-based environment dependency inference for python programs," in *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1245–1256.
[65] C. Wang, R. Wu, H. Song, J. Shu, and G. Li, "smartpip: A smart approach to resolving python dependency conflict issues," in *Proceedings of the 37th International Conference on Automated Software Engineering (ICSE)*, 2022, pp. 1–12.
[66] Y. Wang, M. Wen, Y. Liu, Y. Wang, Z. Li, C. Wang, H. Yu, S.-C. Cheung, C. Xu, and Z. Zhu, "Watchman: Monitoring dependency conflicts for python library ecosystem," in *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 125–135.
[67] Y. Sui and J. Xue, "Value-flow-based demand-driven pointer analysis for c and c++," *IEEE Transactions on Software Engineering*, vol. 46, no. 8, pp. 812–835, 2018.
[68] C. Zhu, R. K. Saha, M. R. Prasad, and S. Khurshid, "Restoring the executability of jupyter notebooks by automatic upgrade of deprecated apis," in *Proceedings of 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 240–252.
[69] A. Vadlamani, R. Kalicheti, and S. Chimalakonda, "Apiscanner-towards automated detection of deprecated apis in python libraries," in *Proceedings of 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 5–8.