Characterizing Deprecated Deep Learning Python APIs: An Empirical Study on TensorFlow

Nian Liu

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

September 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Nian Liu**

Entitled: **Characterizing Deprecated Deep Learning Python APIs: An Empirical Study on TensorFlow**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr.

_____ External Examiner
Dr.

_____ Examiner
Dr.

_____ Supervisor
Dr. Weiyi Shang

Approved by _____
Narayanan, Lata , Chair
Department of Computer Science and Software Engineering

_____ 2021

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Characterizing Deprecated Deep Learning Python APIs: An Empirical Study on TensorFlow

Nian Liu

TensorFlow is a widely used machine learning platform, with millions of people using it to create and train models. It is available in a variety of programming languages, including Python, Java, C++, and JavaScript, among which Python is the most commonly used. Along with TensorFlow's evolution, new Python APIs are introduced, while others may be deprecated. Although the characteristics of deprecated APIs in traditional software frameworks such as Android have been extensively researched in recent years, little attention has been paid to how deprecated APIs in TensorFlow evolve and what impact this has on deep learning. In this thesis, we conducted an empirical study on deprecated Python APIs in TensorFlow. Our study analyzed 20 TensorFlow releases spanning versions 1.0 to 2.3 to investigate API deprecation and its causes. In addition, we studied projects containing 12 popular deep learning models to identify deprecated API usage. Finally, in order to investigate the potential impact of deprecated APIs on deep learning models, we manually updated the deprecated APIs in these projects to compare model accuracy before and after updating. Our research seeks to provide developers with insight into how TensorFlow deprecated APIs evolve, as well as help them understand why APIs became deprecated and the implications of not updating their models by removing deprecated APIs.

Acknowledgments

Throughout the completion of this study, I received lots of support and assistant.

Firstly, I would like to thank my supervisor, Professor Weiyi Shang, who provided insight into the research questions and methodology. Whenever I felt it is difficult to continue and had no idea about the next step work, he would help me analyze the problem and encourage me to try step by step. In addition, there were lots of problems in my two years of study, especially as an international student. He not only guided us on the academic work but helped us to adapt to the local life.

Secondly, I would like to thank my colleagues in my laboratory. They gave me a lot of suggestions in this thesis and provided stimulating discussions about the research. Also, in this pandemic period, life was hard sometimes. We often had a meeting and communication, which made the research completed successfully.

Finally, I would like to thank my parents. I have studied for about 20 years and am just about to graduate now. Without their financial support and accompaniment, I will not be able to complete my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid adoption of Artificial Intelligence has led to the development of many machine learning frameworks to help people build and train machine learning algorithms. TensorFlow is a widely used framework that helps companies such as Airbnb, Intel, and Airbus, to solve real-world, everyday machine learning problems (Community, 2021). TensorFlow provides a collection of APIs that allows developers to create and train machine learning models using mainstream programming languages, such as Python, Java, C++, and JavaScript, and to deploy their projects easily to the cloud, server, or local machine (Abadi et al., 2016).

TensorFlow has released 20 official versions of this writing. Each new version has new features introduced and outdated features suppressed. As TensorFlow has grown rapidly, many APIs are no longer recommended by API designers (i.e., deprecated) since their behavior has become incompatible with the new version. API deprecation in TensorFlow follows a *deprecated-replace-remove* cycle (Li et al., 2020). In this scheme, an API is first flagged as deprecated by inserting a `@deprecated` annotation, indicating that this API is no longer maintained in the framework. The annotation also contains replacement messages describing how to update this deprecated API. Deprecated APIs will be removed after a period of development time in order to clean up the framework.

Prior studies have investigated API evolution and its potential impact on projects. Li et al. (2020) conduct an exploratory study about Android deprecated APIs. They propose an approach to investigate the evolution of deprecated APIs in Android and explored real-world Android apps to study the developers' reaction to deprecation. Zhang et al. (2021) analyze the API evolution in

TensorFlow 2 and disclose reasons for the API evolution. However, their study focuses on API evolution instead of API deprecation, and they do not look into how such evolution affects deep learning models. Some studies have revealed that software evolution might result in compatibility issues. For example, Wei, Liu, and Cheung (2016) discovered that Python framework evolution might lead to compatibility issues, whereas Dietrich, Jezek, and Brada (2014) indicates that the upgrade of Java libraries could cause project problems such as compatibility issues. Unlike traditional software, projects in deep learning focus more on models accuracy instead of performance metrics like CPU usage, response time, etc. To the best of our knowledge, there has not been much focus on how deprecated APIs evolve in TensorFlow and whether their use has an impact on deep learning models. To fill this gap, we conduct an empirical study on characterizing TensorFlow deprecated Python APIs. Specifically, we aim to answer the following research questions:

RQ1: How many TensorFlow APIs are deprecated over time?

RQ2: Why did TensorFlow APIs become deprecated?

RQ3: Are TensorFlow deprecated APIs still used in user projects? If so, to what extend?

RQ4: Do deprecated APIs in deep learning models affect the accuracy of models? If so, how much does accuracy suffer?

In order to answer these research questions, we gather 20 TensorFlow releases ranging from TensorFlow 1.0 to TensorFlow 2.3. RQ1 is then answered by extracting and analyzing the deprecated APIs in each release. For RQ2, we analyze the deprecation message in deprecated APIs to identify the possible reasons for API deprecation. To address RQ3, we also gather 12 popular deep learning models from TensorFlow's official Github organization to explore developers' reactions to these deprecated APIs. To measure the impact of deprecated APIs in deep learning models, we compare the model accuracy in RQ4 before and after updating deprecated APIs. Several findings have been disclosed as a result of these studies: 1) The number of deprecated APIs grows steadily as TensorFlow evolves, especially after TensorFlow 1.13, which represents a significant rise. Only a small portion (4.52%) of deprecated APIs are currently removed, but their survival time is quite short (2 versions in median). Deprecated TensorFlow APIs take a median of 12 versions to become deprecated. 2) Optimization improvements (e.g., **API Optimization** (53.77%)) is the primary reason of Tensorflow API method deprecations. Usability improvements (e.g., **Parameter Name Change**

(91.67%)) is the leading reason for parameter deprecations. 3) Around 5% to 10% of deprecated APIs are still in use even in well-maintained projects. 4) Accuracy difference between deprecated API and their replacements in our test does not appear to be significant. This result indicates that accuracy changes do not incentivize the users of these APIs to migrate to the replacement APIs.

To summarize, this thesis makes the following contributions:

- To the best of our knowledge, this is the first empirical study to reveal the current status of deprecated APIs in TensorFlow.
- We performed the first research discovering the rationale behind deprecated APIs in TensorFlow. We analyzed the deprecation message in 235 deprecated APIs and found 6 API deprecation reasons: 4 at the method level, 2 at the parameter level.
- We automatically uncover deprecated APIs in 12 existing deep learning models, which helps explore developers' reactions to TensorFlow deprecated APIs.
- We present a quantitative study about the impact of deprecated APIs in deep learning models.

Thesis organization. This thesis is organized as follows: Section 2 gives a brief background introduction to help readers better understand this study. Section 3 introduces our experimental setup. Section 4 presents the detailed case study results for four research questions. Section 5 discusses the threats to validity. Section 6 presents related works. Finally, Section 7 concludes this thesis.

Chapter 2

Background

This section provides an overview of TensorFlow Python APIs, API deprecation, and deep learning with TensorFlow.

2.1 TensorFlow Python APIs

TensorFlow (2021) is a machine learning framework that can be used to build and train deep learning models. TensorFlow offers APIs in many popular programming languages, with Python APIs currently being the most popular (Carbonnelle, 2021). These APIs evolve with the release of each TensorFlow version. Since its initial release in 2015, TensorFlow has released 20 versions at the time of this writing, the latest version is version 2.3.

As presented in Figure 2.1, TensorFlow contains multiple abstraction layers. The top layer contains high-level, object-oriented APIs. The second layer from the top is composed of the components for building customized models. The third layer is a low-level API built with Python. Developers can use these APIs to create and implement new machine learning algorithms. The fourth layer is the TensorFlow kernel API which is built with C++. Finally, the bottom-most layer is the hardware layer where TensorFlow code can run on multiple platforms, like CPU, GPU, and TPU.

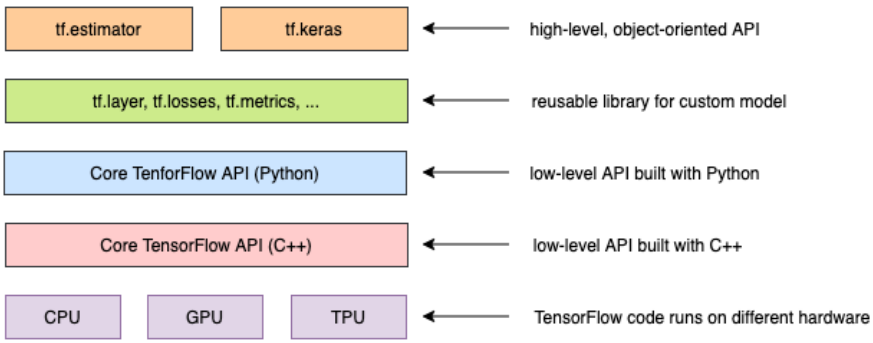

Figure 2.1: Hierarchy of TensorFlow API

2.2 API Deprecation

With the release of new TensorFlow versions, some APIs can not meet the requirement of the framework (e.g., performance or security reason (Li, Bissyandé, Klein, & Le Traon, 2016)), causing APIs to deprecate. However, the deprecated APIs can not be removed directly as they may lead to runtime crashes (Li et al., 2020). In this context, these APIs will first be flagged as deprecated using Python annotations and then removed after a reasonable time (e.g., several releases of the framework).

To mark a deprecated API, developers can wrap it in the `deprecated()` decorator (i.e., `@deprecated`). In this decorator, the arguments passed to `deprecated()` can be used to display a deprecation message, which informs users that an API is deprecated and when it will be removed, as well as what developers can do in the meantime. Listing 2.1 presents a real example of deprecated API in TensorFlow. In this example, `is_gpu_available()` is a deprecated API of test modules. On line 2, API designers notify developers using an alternative API `tf.config.list_physical_devices()`.

```
1 @deprecation.deprecated(None,
2     "Use `tf.config.list_physical_devices('GPU')` instead.")
3 @tf_export("test.is_gpu_available")
4 def is_gpu_available(cuda_only=False, min_cuda_compute_capability=None):
5   """Returns whether TensorFlow can access a GPU.
```

Listing 2.1: An example of deprecated python API in TensorFlow

2.3 Deep Learning with TensorFlow

Deep learning is a research area of Artificial Intelligence (AI) inspired by the human brain and often used to process data-sets and create patterns for use in decision making (Hargrave, 2021). It has been widely used in AI domains, such as image classification, object detection, pattern recognition, and natural language processing (NLP), and proven to be successful (Guo et al., 2016; Lopez & Kalita, 2017; Schmidhuber, 2015).

Listing 2.2 presents an example of the use of TensorFlow API to train and test models. It firstly imports the `mnist` dataset (line 3) and defines the *Sequential* model (line 6) by invoking `tf.keras.models.Sequential()`. Secondly, an optimizer (line 12) and loss function (line 13) are used for model training. Finally, `model.evaluate()` (line 16) is invoked following the invocation of `model.fit()` (line 15) to train and test the model. In this example, the `Keras` module is an open-source software library that acts as an interface for TensorFlow (Keras, 2021).

```
1  """TensorFlow quickstart for beginners"""
2  import tensorflow as tf
3  mnist = tf.keras.datasets.mnist
4  (x_train, y_train), (x_test, y_test) = mnist.load_data()
5  x_train, x_test = x_train / 255.0, x_test / 255.0
6  model = tf.keras.models.Sequential([
7    tf.keras.layers.Flatten(input_shape=(28, 28)),
8    tf.keras.layers.Dense(128, activation='relu'),
9    tf.keras.layers.Dropout(0.2),
10   tf.keras.layers.Dense(10, activation='softmax')
11 ])
12 model.compile(optimizer='adam',
13               loss='sparse_categorical_crossentropy',
14               metrics=['accuracy'])
15 model.fit(x_train, y_train, epochs=5)
16 model.evaluate(x_test,  y_test, verbose=2)
```

Listing 2.2: Example of model built with TensorFlow

Chapter 3

Experimental Setup

In this section, we discuss our experimental setup.

This study aims to analyze deprecated Python APIs in TensorFlow and their potential impact on deep learning models. The experimental data consists of two parts: 1) 20 TensorFlow releases spanning version 1.0 to 2.3 extracted from TensorFlow's official Github website [1]. 2) 12 Deep Learning models maintained by the official TensorFlow Github organization.

In the following sections, we introduce these two parts of the dataset.

3.1 TensorFlow versions

TensorFlow is an open-source Machine Learning framework that was first released in 2015. It is available on Github with ∼95K commits, ∼3K contributors, and 49 branches at the time of this study.

To answer RQ1 and RQ2, we collect the source code of TensorFlow releases from the official TensorFlow Github repository. Table 3.1 displays a collection of 20 Github branches spanning TensorFlow versions 1.0 to 2.3. Each branch corresponds to a TensorFlow version examined in our study. We explore each version, including the most recent source code update at the time of this study.

[1] https://github.com/tensorflow/tensorflow

Table 3.1: Overview of studied TensorFlow branches

TF version	Branch	HEAD Date
2.3	r2.3	2020-07-24
2.2	r2.2	2020-07-24
2.1	r2.1	2020-07-24
2.0	r2.0	2020-07-24
1.15	r1.15	2020-07-24
1.14	r1.14	2019-08-22
1.13	r1.13	2019-07-15
1.12	r1.12	2019-06-21
1.11	r1.11	2018-09-25
1.10	r1.10	2018-08-23
1.9	r1.9	2018-07-09
1.8	r1.8	2018-06-04
1.7	r1.7	2018-05-07
1.6	r1.6	2018-05-30
1.5	r1.5	2018-03-27
1.4	r1.4	2018-07-09
1.3	r1.3	2017-10-17
1.2	r1.2	2018-03-22
1.1	r1.1	2017-05-18
1.0	r1.0	2017-05-02

3.2 Deep Learning models

To address RQ3 and RQ4, we gather the source code of deep learning models in order to investigate the use of deprecated APIs and their impact on deep learning models. These deep learning models are extracted from a repository from offficial TensorFlow Github website: *Models*[2]. The *Models* repository stores various implementations of state-of-the-art (SOTA) deep learning models and modelling solutions for TensorFlow users (Yu et al., 2020). The root directory of *Models* includes three sub-directories: official, research, and community. The official part is maintained by the official TensorFlow development team and uses the latest APIs, while the research and community parts are maintained by developers in the TensorFlow community, where APIs may have become outdated. Therefore, in this study, we choose the official part as our research focus. Table 3.2 displays the Github branches of *Models* analyzed for our study. Each branch is associated with

[2] `https://github.com/tensorflow/models`

Table 3.2: Overview of studied *Models* branches. There are no releases for TensoFlow 1.14 and 1.15

API version	Branch	HEAD Date
2.3	r2.3	2020-07-30
2.2	r2.2	2020-04-15
2.1	r2.1	2020-01-07
2.0	r2.0	2020-10-15
1.13	r1.13	2019-02-06
1.12	r1.12	2018-11-07
1.11	r1.11	2018-08-31
1.10	r1.10	2018-07-16

one version of TensorFlow API. It should be noted that there is no release of studied *Models* for TensorFlow 1.14 and 1.15.

Chapter 4

Case Study Results

In this section, we answer the following four research questions.

4.1 RQ1: How many TensorFlow APIs are deprecated over time?

Motivation

TensorFlow has experienced a long development period. Since its first release in 2015, around 20 official versions were released until the time of this research. With the increased focus on freshly introduced APIs in TensorFlow, the evolution of deprecated APIs has been the focus of few studies. However, investigating the status of API deprecation in TensorFlow could help developers better understand the evolution patterns of TensorFlow and prepare for the migration of the deprecated APIs in their projects.

Approach

Table 3.1 presents the TensorFlow branches selected for our study. We analyzed 20 TensorFlow branches ranging from TensorFlow 1.0 to TensorFlow 2.3. We also examined the most current source code update at the time of the study and manually cloned the repository to our own machine.

To answer RQ1, we first identify the deprecated Python APIs in the studied TensorFlow versions. An API could be tagged as deprecated in both documentation and source code. Li et al.

(2020) disclose there is an inconsistency in tagging depreciated APIs between source code and documentation for Android projects. Their study finds that some deprecated APIs are highlighted in the documentation but have been removed in the source code. Therefore, to make our study results more accurate, we use TensorFlow source code as our study focus rather than TensorFlow documentation.

The Abstract Syntax Tree (AST) is a potent tool of the Python programming language and provides tree structure for source code that can be used for static analysis of Python projects. For each Python file, we build an AST and traverse all of the public classes and methods. For this RQ, we use AST to retrieve useful information (i.e., API name, annotation) for deprecated APIs at the class and method levels. As mentioned in section 2, deprecated APIs are identified by the decorator `@deprecated`. Thus, if a class or method has such a decorator, it is considered deprecated and added to a list. Furthermore, if an API was deprecated in the previous version and disappears in the current version, we consider it removed; if an API was not deprecated in the previous version but becomes deprecated in the current version, we consider it newly deprecated.

Results

In this section, we present the results of RQ1 through three dimensions: ***Deprecated APIs***, ***Deprecated APIs and retained APIs*** and ***Survival time***.

Deprecated APIs. In this dimension, we study the distribution of deprecated APIs in TensorFlow's historical and latest releases.

Table 4.1 provides an overview of studied TensorFlow versions. The column **TF Version** depicts the TensorFlow version that we analyzed. The column **SLOC** displays the number of source lines of code, while the column **APIs** displays the total number of APIs for each TensorFlow version. The column **Deprecated APIs** is the number of deprecated APIs. According to this table, as TensorFlow matures, SLOC, APIs, and Deprecated APIs increase while deprecated APIs increase more. Specifically, from TensorFlow 1.0 to TensorFlow 2.3, SLOC has increased five-fold, while the number of APIs has increased four-fold. The number of deprecated APIs has increased from 13 to 190 (i.e., nearly 15 times).

Figure 4.1 presents the distribution of deprecated API rate (the number of deprecated APIs to

Table 4.1: An overview of studied TensorFlow versions. Deprecated APIs are considered as long as they are annotated

TF Version	SLOC	APIs	Deprecated APIs
2.3	412K	6,586	190
2.2	386K	6,251	166
2.1	355K	5,934	155
2.0	337K	5,714	150
1.15	338K	5,707	150
1.14	314K	5,338	132
1.13	257K	4,419	116
1.12	229K	3,786	45
1.11	211K	3,470	29
1.10	206K	3,357	23
1.9	195K	3,251	22
1.8	188K	3,172	22
1.7	182K	3,033	22
1.6	176K	2,948	22
1.5	167K	2,798	21
1.4	154K	2,612	15
1.3	113K	2,002	15
1.2	106K	1,888	13
1.1	90K	1,659	13
1.0	82K	1,525	13

total APIs) in each Tensorflow version. The result shows that in TensorFlow's early development stage, from 1.0 to 1.12, the rate remains mostly stable despite the growth in the number of deprecated APIs. Between versions 1.12 and 1.13, the deprecated API rate has a huge increase from 1.19% to 2.63%. Indeed, as illustrated in Table 4.1, the number of deprecated APIs has almost tripled in 1.13 compared to version 1.12, but the total number of deprecated APIs only increases slightly. After TensorFlow 1.13, the rate of deprecated APIs to total APIs is steady again.

Deprecated APIs and retained APIs. In this dimension, we look into whether the deprecated APIs are removed from TensorFlow.

Figure 4.2 presents the number of total deprecated APIs (in accumulation) and retained deprecated APIs. We can see that both polylines rise and are fairly near, implying that most deprecated APIs are still available in the framework even though they have been deprecated. Only a minor proportion of deprecated APIs are removed (9 out of 199, 4.52%).

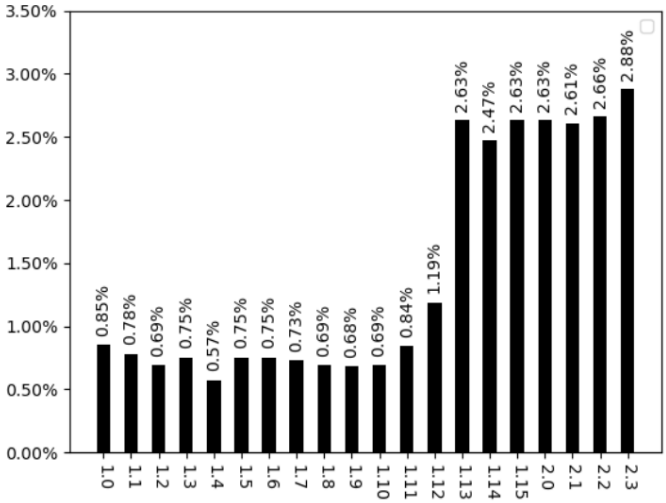

Figure 4.1: Distribution of deprecated API rate (the number of deprecated APIs to total APIs). All deprecated APIs, including those that were deprecated in prior versions, are evaluated for each API version

Table 4.2: The number of added and removed deprecated APIs for each update

TF Upgrade	Addition	Removal
1.0 - 1.1	0	0
1.1 - 1.2	0	0
1.2 - 1.3	2	0
1.3 - 1.4	0	0
1.4 - 1.5	8	2
1.5 - 1.6	1	0
1.6 - 1.7	0	0
1.7 - 1.8	0	0
1.8 - 1.9	0	0
1.9 - 1.10	1	0
1.10 - 1.11	6	0
1.11 - 1.12	16	0
1.12 - 1.13	72	1
1.13 - 1.14	18	2
1.14 - 1.15	18	0
1.15 - 2.0	1	1
2.0 - 2.1	8	3
2.1 - 2.2	11	0
2.2 - 2.3	24	0

Table 4.2 provides a summary of the addition and removal of deprecated APIs for each TensorFlow version upgrade. From TensorFlow 1.0 to 1.9, 3 out of 9 version upgrades add a few

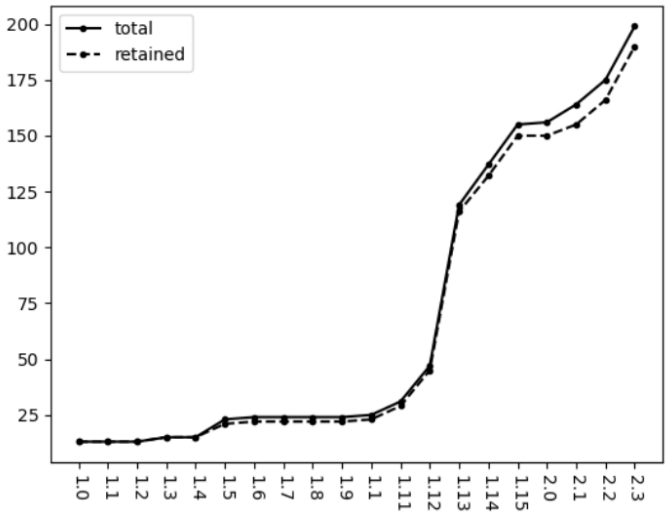

Figure 4.2: The total number of deprecated APIs (in accumulation) and retained deprecated APIs

deprecated APIs, and only one upgrade removes two deprecated APIs. Since TensorFlow 1.9, several APIs are deprecated in each upgrade, very few deprecated APIs are removed. TensorFlow 1.12 to TensorFlow 1.13 has the most number of deprecation addition.

Survival time. In this dimension, we examine the survival time of APIs, including the survival time of deprecated APIs and lifespan of APIs.

Figure 4.3 is a density plot illustrating the distribution of deprecated APIs' survival time. We model the survival time as the number of TensorFlow versions a deprecated API is removed from TensorFlow since it is deprecated. There are 9 deprecated APIs removed from TensorFlow with a median age of two releases, i.e., for those removed deprecated APIs, each of them gets removed after a median of two releases. Despite the fact that only a few APIs are removed, the survival time is rather short. According to this figure, two APIs are removed in one version after being deprecated, and no API survives more than two TensorFlow releases. Considering the quick release of TensorFlow (i.e., in Table 3.1, 20 versions released in 39 months, averaging 2 months for one version), developers need to migrate their code quickly before these deprecated APIs become inaccessible.

Then we examine the lifespan of APIs, i.e., the number of TensorFlow versions since APIs were introduced until they become deprecated.

Figure 4.4 depicts a density plot of the distribution of lifespans for retained APIs and removed

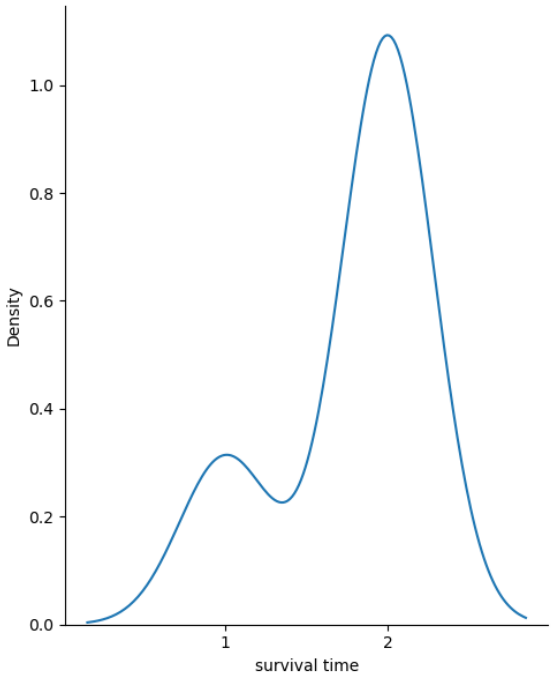

Figure 4.3: Density distribution of the survival time of TensorFlow deprecated APIs. Survival time corresponds to the number of versions a deprecated API is removed from TensorFlow since it is deprecated

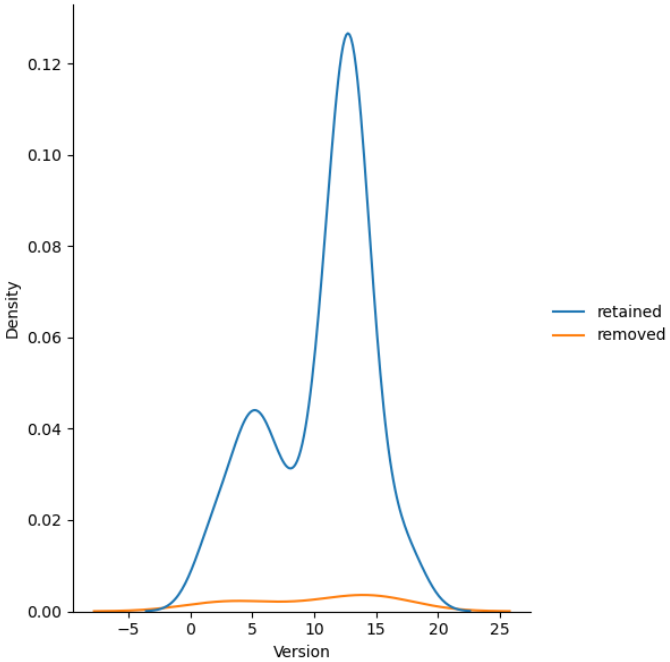

Figure 4.4: Density distribution of the number of versions TensorFlow APIs (between retained and removed deprecated APIs) get deprecated since introduced into TensorFlow

APIs. The distribution of retained and removed APIs does not have a significant difference. In general, it takes a median of 12 TensorFlow versions of retained APIs to become deprecated and 13 versions of removed APIs to become deprecated. Most deprecated APIs take around 13 versions to get deprecated, which gives developers a comparatively long period to adopt these APIs and do not need to update their code frequently.

Li et al. (2020) did an exploratory study about API deprecation in Android. They found that 16% deprecated APIs are removed in Android while this number is only 4.52% in TensorFlow. The survival time of deprecated APIs in Android and TensorFlow are both short. Most deprecated APIs take a comparatively long period to get deprecated In TensorFlow, whereas in Android, removed APIs get deprecated quickly than retained APIs.

In summary, the number of deprecated APIs in TensorFlow grows steadily while TensorFlow 1.13 has a significant increase. Only a small proportion (4.52%) of deprecated APIs are removed eventually, but their survival time is relatively short (2 versions in median). Most deprecated APIs take a comparatively long period (12 versions in median) to get deprecated since they were introduced into TensorFlow.

RQ1 Finding

Deprecated APIs increase constantly as TensorFlow evolves, with TensorFlow 1.13 seeing a significant increase. Only a small proportion (4.52%) of deprecated APIs are eventually removed, but their survival time is quite short (2 versions in median). Most deprecated APIs take a comparatively long period (12 versions in median) to get deprecated since they were introduced into TensorFlow.

4.2 RQ2: Why did TensorFlow APIs become deprecated?

Motivation

In this section, we explore TensorFlow's deprecated APIs to discover the reasons for their deprecation. In RQ1, after TensorFlow 1.9, almost every TensorFlow upgrade will add some new deprecated APIs; this motivates us to investigate why these APIs become deprecated. The answers to this research question can provide insight into the evolution of the Tensorflow API.

Approach

The following is our approach workflow.

As discussed in RQ1, we extract useful messages for deprecated APIs in the latest TensorFlow version (2.3), including API names, fields, annotations, etc. For each deprecated API, we first explore the deprecation message extracted by AST to discover rationales of API deprecation. If the deprecation message is missing or inaccurate, we manually examine the API source code. We then conduct pair reviews (i.e., two persons working on reviewing individually) on all deprecated APIs to label their deprecation rationales and reached an agreement of 88.9%. We then merge our results after discussion with a third reviewer to eliminate bias in the results.

Results

In this part, we present the results of RQ2.

In this study, we analyzed 235 deprecated APIs in TensorFlow 2.3 (including those removed in the previous versions).

Table 4.3: API Deprecation Rationales

Deprecation Rationale	Level
API Optimization	Method
API Name Change	Method
Compatibility Issue	Method
Feature Deleting	Method
Parameter Name Change	Parameter
Unnecessary Parameter	Parameter

Deprecation rationale classification. Based on the results of our analysis, we identified six API deprecated rationales, four of which are at the method level and two at the parameter level, as presented in Table 4.3. The following is an introduction of these six rationales:

1) API Optimization: As TensorFlow evolves, many older APIs are no longer able to meet its increasing requirements. Developers introduce newer APIs to replace the older ones, bringing many advantages, such as improved performance, organization, etc.

```
1 @deprecation.deprecated(date=None, instructions="Use `tf.cast` instead.")
2 @tf_export(v1=["to_float"])
3 @dispatch.add_dispatch_support
4 def to_float(x, name="ToFloat"):
5   """Casts a tensor to type `float32`."""
6   return cast(x, dtypes.float32, name=name)
```

Listing 4.1: An example of API deprecation due to **API Optimization** from *math_ops.py* at TensorFlow 2.3

Listing 4.1 provides an example of this rationale. `to_float()` (line 4) is a Python API used to cast a tensor to type **float32** where tensor is a multidimensional array. Actually, there are some other APIs with similar functionality, such as `to_int32()`, `to_double()`, etc. These APIs can be replaced a single new API `tf.cast()`. This type of API change merges multiple APIs to one API and makes the API calling more concise and organized.

2) API Name Change: In this rationale, the replaced APIs are given new names without changing their functionality or the underlying source code. As TensorFlow evolves, API names may not reflect the current functionality, so they are changed to meet the new requirements.

```
1 @deprecation.deprecated(
2     date=None, instructions='Please use `layer.add_weight` method instead.')
3 @doc_controls.do_not_doc_inheritable
4 def add_variable(self, *args, **kwargs):
5     """Deprecated, do NOT use! Alias for `add_weight`."""
6     return self.add_weight(*args, **kwargs)
```

Listing 4.2: An example of API deprecation due to **API Name Change** from *base_layer.py* at TensorFlow 2.3

Listing 4.2 presents an example of this rationale. The API `add_variable()` (line 4) allows developers to add variables to deep learning layers, which is an alias for `add_weight()` (line 5). However, developers discovered that `add_weight()` is more appropriate as the variables in deep learning are officially termed **weight**, hence `add_variable()` became deprecated.

3) Compatibility Issue: TensorFlow releases a new version every few months. A new version typically introduces new features which may cause compatibility issues for older APIs, preventing them from being used in the newer version.

```
1 @deprecation.deprecated(
2     None,
3     "This function will only be available through the v1 compatibility "
4     "library as tf.compat.v1.saved_model.utils.build_tensor_info or "
5     "tf.compat.v1.saved_model.build_tensor_info.")
6 def build_tensor_info(tensor):
7   """Utility function to build TensorInfo proto from a Tensor.
```

Listing 4.3: An example of API deprecation due to **Compatibility Issue** from *utils_impl.py* at TensorFlow 2.3

Listing 4.3 shows an example of an API that is deprecated due to compatibility issues. As TensorFlow evolves, some older APIs are no longer compatible with the newer version of TensorFlow, such as `build_tensor_info()` (line 6). The docstring (line 3–5) of this deprecated API indicates that it will only be available in TensorFlow 1.x.

4) Feature Deleting: TensorFlow adds or deletes features in each new release. When a feature is removed from a version, the relevant API may become deprecated.

```
1 # We no longer track graph in tf.layers layers. This property is only kept to
    maintain API backward compatibility.
2 @property
3 @deprecation.deprecated(
4     date=None,
5     instructions='Stop using this property because tf.layers layers no longer
    track their graph.')
6 def graph(self):
7     if context.executing_eagerly():
```

```
8         raise RuntimeError('Layer.graph not supported when executing eagerly.')
9     return None
```

Listing 4.4: An example of API deprecation due to **Feature Deleting** from *base.py* at TensorFlow 2.3

Listing 4.4 presents an example of this rationale. Since the `instruction` property (line 5) of the API indicates that the graph is no longer tracked by `tf.layers`, this API is now deprecated.

5) Parameter Name Change: Similar to API name changes, Parameter names may also be updated. The new parameter names attempt to be more intuitive and understandable than the old ones, making the APIs more user-friendly.

```
1 @deprecation.deprecated_args(None, "Please use `rate` instead of `keep_prob`. ""
    Rate should be set to `rate = 1 - keep_prob`.", "keep_prob")
2 def dropout(x, keep_prob=None, noise_shape=None, seed=None, name=None,
3             rate=None):
4   """Computes dropout.
```

Listing 4.5: An example of parameter deprecation due to **Parameter Name Change** from *builder_impl.py* at TensorFlow 2.3

Listing 4.5 presents an example with the API's parameter name changed. The API `dropout()` (line 2) has a parameter named `keep_prob`. However, developers believe that `rate` is more appropriate than `keep_prob` because it is more commonly used, therefore `keep_prob` has been deprecated and replaced with `rate`.

6) Unnecessary Parameter: Some API parameters could also be deprecated, in addition to the API itself. With the evolution of APIs, their parameters may become unnecessary due to the changes in API logic. Parameter deprecation is discussed in this rationale.

```
1 @deprecation.deprecated_args(None, '`inputs` is now automatically inferred', '
    inputs')
2 @doc_controls.for_subclass_implementers
3 def add_update(self, updates, inputs=None):
4     """Add update op(s), potentially dependent on layer inputs.
```

Listing 4.6: An example of parameter deprecation due to **Unnecessary Parameter** from *base_layer.py* at TensorFlow 2.3

Listing 4.6 illustrates an example of this rationale. The API `add_update()` (line 3) has a parameter `inputs` for developers to manually provide inputs, but `inputs` is now automatically inferred according to the argument at line 1. Therefore, this parameter will no longer be required for manual input by developers.

Deprecation rationale analysis. By performing a pair review of these 235 APIs, we reached an agreement for the initial classification of 209 APIs (88.9%). As for the remaining 26 APIs, we integrated our discussion results after discussing them with a third reviewer. Figure 4.5 and Figure 4.6 present the rationales of API deprecation. More specifically, Figure 4.5 depicts the method level (199 out of 235) of API deprecation rationales, whereas Figure 4.6 depicts the parameter level (36 out of 235).

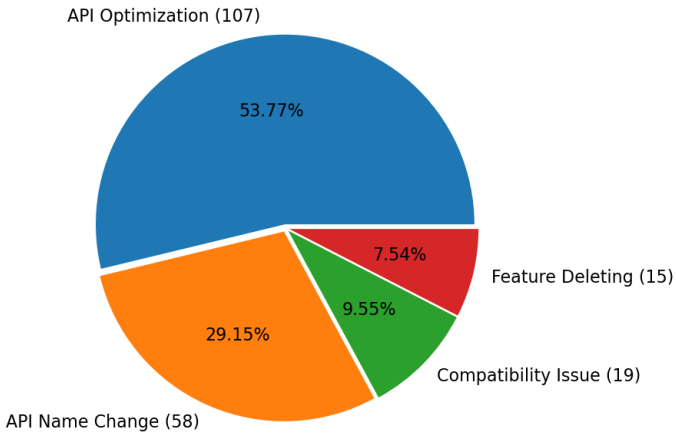

Figure 4.5: Distribution of method-level API deprecation rationales. The number of deprecated APIs for each rationale is indicated in parentheses

As indicated in Figure 4.5, **API Optimization** is the leading cause of API deprecation, taking up 53.77% (107). **API Name Change** accounts for 29.15% (58) of all deprecated APIs at the method level. This rationale includes APIs with merely a name change but the same functionality. By renaming the API, the framework could be more organized and user-friendly. **Feature Deleting** has the fewest deprecated APIs at the method level, with 7.54% (15), which indicates that TensorFlow is

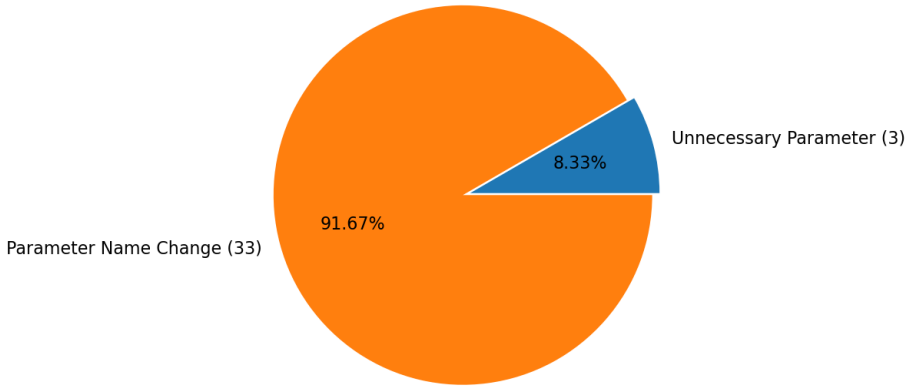

Figure 4.6: Distribution of parameter-level API deprecation rationales. The number of deprecated APIs for each rationale is indicated in parentheses

still focusing on introducing new features rather than removing the existing features, as highlighted by RQ1 Finding. There are 19 APIs (9.55%) that fall into the **Compatibility Issue**. Such APIs are only available in TensorFlow 1.x and are likely to be removed in upcoming TensorFlow releases.

Figure 4.6 presents the rationales of API deprecation at the parameter level. Among the 36 deprecated APIs, 33 (91.67%) are deprecated due to **Parameter Name Change**, implying that **Parameter Name Change** is the most common rationale for API deprecation at the parameter level. Only 3 APIs (8.33%) fall into the **Unnecessary Parameter**.

Sawant, Huang, Vilen, Stojkovski, and Bacchelli (2018) analyzed 4 Java frameworks and found that introducing a new feature and functional defect are the most frequent reasons for deprecating an API in Java. This finding is similar to our result where **API Optimization** being the most common rationale. However, 29.15% APIs are deprecated because of **API Name Change** in TensorFlow, while renaming of feature only takes a very small portion in Java.

In conclusion, **API Optimization** is the most prevalent rationale for API deprecation at the method level, accounting for 53.77%. Moreover, **API Name Change** (29.15%) is the second most common rationale, followed by **Compatibility Issue**(9.55%). **Parameter Name Change** (91.67%) is the most leading deprecation rationale at parameter level.

RQ2 Finding

Tensorflow API method deprecations are lead by optimization improvements (e.g., ***API Optimization*** *(53.77%)), meanwhile, parameter deprecations are lead by usability improvements (e.g.,* ***Parameter Name Change*** *(91.67%)).*

4.3 RQ3: Are TensorFlow deprecated APIs still used in user projects, if so, to what extent?

Motivation

In TensorFlow, deprecation follows a *deprecated-replace-remove* cycle, as discussed in section 1. However, replacing deprecated APIs may take a long time because developers encounter challenges such as becoming accustomed to using deprecated APIs or deprecated APIs having no substitute message provided (Li et al., 2020). In this section, we examine the use of deprecated APIs with some popular deep learning models. Through this research question, we would like to discover if and to what extent deprecated APIs are used in TensorFlow projects.

Approach

The following is our approach workflow.

Data extration. Our study draws on the *Models* repository from official TensorFlow Github website which consists of several deep learning models, as indicated in Section 3.

Table 4.4 presents the statistics of analyzed *Models* branches. There are eight *Models* branches, each of which corresponds to a different TensorFlow API version from 1.10 to 2.3 (however, TensorFlow 1.14 and 1.15 are not available). From branch `r1.10` (the earliest branch) to `r2.3` (the most recent branch at the time of the study), the source lines of code (SLOC) have increased by six times, and the number of methods has increased by more than four times. Furthermore, the number of deep learning models in this repository has increased from 5 to 12.

Now we take branch `r2.3` as an example and display the deep learning models for this branch in Table 4.5. As we can see, there are 12 deep learning models implemented in this repository. These models are divided into categories that include a variety of deep learning areas such as Image Classification (IC), Natural Language Processing (NLP), etc.

Deprecated API search. First, we clone the *Models* repository from Github to our local machine. For each repository branch, we determine which TensorFlow API version it is using, so that we

Table 4.4: Statistic overview of selected *Models* branches

Branch	SLOC	Methods	Models	API Version
r2.3	53K	939	12	2.3
r2.2	51K	956	9	2.2
r2.1	47K	848	9	2.1
r2.0	31K	638	8	2.0
r1.13	12K	285	5	1.13
r1.12	11K	254	5	1.12
r1.11	10K	227	5	1.11
r1.10	9K	218	5	1.10

Table 4.5: Deep learning models in *Models* r2.3

Model	Category	Designers
EfficientNet	Image Classification	Lin, Goyal, Girshick, He, and Dollár (2017)
Mnist	Image Classification	LeCun, Bottou, Bengio, and Haffner (1998)
ResNet	Image Classification	K. He, Zhang, Ren, and Sun (2016)
Mask R-CNN	Object Detection and Segmentation	K. He, Gkioxari, Dollár, and Girshick (2017)
RetinaNet	Object Detection and Segmentation	Lin et al. (2017)
ShapeMask	Object Detection and Segmentation	Kuo, Angelova, Malik, and Lin (2019)
ALBERT	Natural Language Processing	Lan et al. (2019)
BERT	Natural Language Processing	Devlin, Chang, Lee, and Toutanova (2018)
NHNet	Natural Language Processing	Gu et al. (2020)
Transformer	Natural Language Processing	Vaswani et al. (2017)
XLNet	Natural Language Processing	Yang et al. (2019)
NCF	Recommendation	X. He et al. (2017)

can know what deprecated APIs it may be using according to the deprecated API list collected in RQ1. We then look for keywords associated with these deprecated APIs in the source code of *Models*. We also add certain filter conditions to reduce the number of false positives. For example, for the deprecated API `conv2d()`, we search `.conv2d` to ensure that it is actually an API invocation. However, because several Python libraries may have the same API name, there may be some false positives. As a result, we perform a manual check to filter out false positives after extracting all of the deprecated API used in *Models*.

Results

In this section, we present the results of RQ3.

Table 4.6 indicates the distribution of deprecated APIs in *Models*. Column **Dep APIs** displays

Table 4.6: Distribution of deprecated APIs in *Models*

Branch	Mnist	Res[1]	Reti[2]	Trans[3]	XLN[4]	Oth[5]	Dep APIs	Dep APIs (TF)	Per (%)
r2.3	0	1	2	4	2	1	10	190	5.26
r2.2	2	4	1	4	3	0	14	166	8.43
r2.1	2	4	1	4	3	0	14	155	9.03
r2.0	1	5	0	4	2	0	12	150	8.00
r1.13	1	5	0	3	0	1	9	116	7.76
r1.12	0	0	0	0	0	0	0	45	0
r1.11	0	0	0	0	0	0	0	29	0
r1.10	0	0	0	0	0	0	0	23	0

1: Res is ResNet.
2: Reti is RetinaNet.
3: Trans is Transformer.
4: XLN is XLNet.
5: Oth is Others, it includes files not belong to any particular model.

the number of deprecated APIs used in *Models*, whereas column **Dep APIs (TF)** displays the total number of deprecated APIs in the corresponding TensorFlow version. Column **Per (%)** displays the percent of the number of deprecated APIs used in *Models* to that in the corresponding TensorFlow version. We could see that in *Models* `r2.3`, there are 10 deprecated APIs used, which is 5.26% of the total deprecated APIs in TensorFlow 2.3. A similar proportion of deprecated APIs exists in versions 2.3, 2.2, 2.1, 2.0, and 1.3 of the *Models*. The result indicates that there are still some deprecated APIs in *Models* even though TensorFlow does not recommend using them. *Models* versions `r1.12`, `r1.11.0`, and `r1.10.0` do not have any deprecated APIs, which could be due to the fact that these branches have fewer source lines of code, and the total number of deprecated APIs before TensorFlow 1.13 is lower, as discussed in RQ1.

Table 4.7 presents the frequency of deprecated APIs used in *Models*. The most frequent API is `tf.to_float()`, which appears 61 times followed by `tf.to_int32()` appears 50 times. `tf.data.experimental.parallel_interleave` (24) and `tf.py_func` (18) both have a frequency more than 15 times. In *Models*, 16 of the 17 deprecated APIs appear twice or more. *Models* is well maintained by TensorFlow official developers, and employs the latest up-to-date APIs. However, there are still 203 deprecated API calls in this project, not to mention third-party projects that may not be able to update their TensorFlow releases in time. This result is similar with what found in Android, where Li et al. (2020) indicates that 61.97% Android apps are making use of deprecated APIs. Although

Table 4.7: Frequence of deprecated APIs being used in *Models*

API	Frequency
tf.to_float	61
tf.to_int32	50
tf.data.experimental.parallel_interleave	24
tf.py_func	18
tf.sparse_to_dense	9
tf.data.make_one_shot_iterator	6
tf.io.tf_record_iterator	5
tf.keras.backend.set_learning_phase	5
tf.nn.softmax_cross_entropy_with_logits	4
tf.layers.batch_normalization	4
tf.layers.conv2d	4
tf.layers.dense	4
tf.layers.max_pooling2d	4
tf.data.experimental.map_and_batch	3
tf.test.is_gpu_available	2
tf.test.TestCase.test_session	2
tf.config.experimental_run_functions_eagerly	1
Total	203

most of these APIs appear easy to migrate, the developers are not doing so. To investigate the advantages or disadvantages of migrating these APIs, we manually update them and observe the effect in model accuracy in RQ4.

RQ3 Finding

Despite the fact that Models is well-maintained by TensorFlow official, it still has deprecated APIs. There was no deprecated API in Models prior to TensorFlow 1.13. However, following the release of TensorFlow 1.13, deprecated APIs ($\sim$ 5-10 % of TensorFlow deprecated APIs) have appeared, which could be attributed to the significant increase in deprecated APIs in TensorFlow 1.13.

4.4 RQ4: Do deprecated APIs in deep learning models affect the accuracy of models? If so, how much does accuracy suffer?

Motivation

According to RQ3, even in projects maintained by TensorFlow official like *Models*, there are still deprecated APIs. To the best of our knowledge, no prior research has looked into how deprecated APIs in deep learning projects affect model accuracy. To fill this gap, we manually upgrade the deprecated APIs found in RQ3 and compare the model accuracy difference before and after updating to investigate the potential impact of deprecated APIs on deep learning models.

Approach

To answer RQ4, we first identify all deep learning models that use all of the deprecated APIs (17) collected in RQ3. Then we train and test models 20 times with their original parameters. Following that, we update deprecated API invocations for one deprecated API based on replacement messages from its annotation and documentation, then repeat the training and testing a further 20 times. In this way, we can collect the model accuracy before and after deprecated API updating, yielding 40 result records (i.e., 20 before and 20 after) for each deprecated API. We follow the same procedure for all deprecated APIs. Finally, we compute the *p*-value and *Cohen's D* through the *t*-test API in `pingouin` library between the results before and after deprecated API update for each deprecated API to look into the impact of deprecated APIs on deep learning models. `pingouin` (Vallat, 2018) is an open source python package used to perform statistical computation.

Among all the 17 distinct deprecated APIs used in *Models*, 5 APIs obviously have no impact on model accuracy and are therefore excluded from our experiment:

- `tf.data.make_one_shot_iterator`, `tf.io.tf_record_iterator`: designed to iterate over training data, without resolving value computations.
- `tf.test.is_gpu_available`: a method for displaying GPU information that has no impact on model accuracy.

Table 4.8: Deprecated API and its replacement

Deprecated API	Replaced API
tf.to_float	tf.cast
tf.to_int32	tf.cast
tf.data.experimental.parallel_interleave	tf.data.Dataset.interleave
tf.py_func	tf.py_function
tf.sparse_to_dense	tf.sparse.to_dense
tf.nn.softmax_cross_entropy_with_logits	tf.nn.softmax_cross_entropy_with_logits_v2
tf.layers.batch_normalization	tf.keras.layers.BatchNormalization
tf.layers.conv2d	tf.keras.layers.Conv2D
tf.layers.dense	tf.keras.layers.Dense
tf.layers.max_pooling2d	tf.keras.layers.MaxPool2D
tf.data.experimental.map_and_batch	tf.data.Dataset.map; tf.data.Dataset.batch
tf.config.experimental_run_functions_eagerly	tf.config.run_functions_eagerly

- `tf.keras.backend.set_learning_phase`: used to specify whether the current phase is training or testing.
- `tf.test.TestCase.test_session`: called in a file located in the *official/utils/testing* folder, which normally contains helper files and is unrelated to any particular model.

For the remaining 12 deprecated APIs, we manually updated them following the replacement messages. Table 4.8 shows the deprecated API and its replacement.

Table 4.9: Deep Learning Models and used Datasets

Models	Dataset	Training Steps	Hardware
Minst	Minst (LeCun et al., 1998)	24,000	Tesla V100
Transformer	WMT17(Bojar et al., 2017)	5,000	Tesla V100
Resnet	Cifar-10 (Krizhevsky, 2009)	390	Google v3-8
Resnet	ImageNet (Russakovsky et al., 2015)	5,000	Google v3-8
XLnet	IMDB (Maas et al., 2011)	2,000	Google v3-8
RetinaNet	COCO (Lin et al., 2014)	5,000	Google v3-8

In RQ3, we found deprecated APIs in 5 deep learning models. Table 4.9 shows the Deep Learning Models and used Datasets. We trained and tested these models on a Tesla V100 GPU and Google v3-8 TPU, and our deep learning models are trained and tested using the same settings listed in documentation except for the training steps. Ideally, these deep learning models should be trained as many times as possible to achieve satisfying accuracy, which may take a long period. However,

because our research focuses on differences in accuracies before and after deprecated API updates rather than predicting the dataset, a large training step number is unnecessary.

p-value is a statistical measure for indicating the statistical significance of relationships between two data groups (Thiese, Ronna, & Ott, 2016). It is a probability that indicates whether the null hypothesis (i.e., assuming no difference between the two groups of data samples) is rejected or not. If the *p*-value is less than a certain threshold (typically 0.05), the null hypothesis is rejected, implying that there is a difference between the two data samples. In this study, our null hypothesis is that model accuracy before and after deprecated API updating remains the same, therefore if the *p*-value is less than the threshold (0.05), the model accuracy has a difference; otherwise, it does not.

Cohen's D is an effect size used to indicate the magnitude of the difference between data groups (Sullivan & Feinn, 2012). As the effect size grows, the relationship between two data groups gets stronger. *Cohen's D* of 0.2 is considered as a small effect, 0.5 is medium, and 0.8 is large. *Cohen's D* is independent of sample size, but *p*-value will be influenced by sample size, while with a sufficiently large sample, the *p*-value will almost always show significant (Sullivan & Feinn, 2012). Therefore, In this study, we use *Cohen's D* combined with *p*-value to measure the model accuracy difference before and after deprecated API updates.

t-test is a type of statistical test that is used to compare the means of two groups (Yim, Nahm, Han, & Park, 2010). Two sample *t*-test consists of independent samples and paired samples. Independent samples are independent of each other, while paired samples are dependent on each other. The model accuracies before and after deprecated API updates are two independent samples. Therefore, In our test, we used the independent *t*-test to evaluate the model accuracy difference before and after updating the deprecated APIs.

Results

In this section, we present the results of RQ4.

We trained and tested the associated deep learning models for the 12 deprecated APIs, and the results are included in Table 4.10. Column **API Name** displays the names of deprecated APIs, and their full-qualified names can be found in Table 4.7. Column **Version** is the TensorFlow API version for each deprecated API. Column **Model** displays the deep learning model that encloses

Table 4.10: Difference in accuracy of deep learning models before and after deprecated API updates

API Name	Version	Model	Train	Acc (B)	Acc (A)	*p*-val	*Cohen's D*
`to_int32`	1.13.0	Mnist	24,000	99.26%	99.27%	0.59	0.17 (S)
`soft`[1]	2.2.0	Trans[8]	5,000	4.59	4.63	0.56	0.18 (S)
`to_int32`	1.13.0	Trans	5,000	4.94	4.89	0.77	0.11 (S)
`to_float`	1.13.0	Trans	5,000	4.94	4.79	0.34	0.39 (M)
`sparse`[2]	2.1.0	ResNet	390	16.13%	16.67%	0.75	0.10 (S)
`exp`[3]	2.3.0	ResNet	5,000	0.19%	0.22%	0.32	0.35 (M)
`conv2d`	1.13.0	ResNet	5,000	0.16%	0.17%	0.79	0.08 (S)
`max_`[4]	1.13.0	ResNet	5,000	0.16%	0.17%	0.86	0.06 (S)
`dense`	1.13.0	ResNet	5,000	0.16%	0.21%	0.14	0.47 (M)
`batch`[5]	1.13.0	ResNet	5,000	0.16%	0.21%	0.17	0.44 (M)
`map_`[6]	2.1.0	XLnet	2,000	90.02%	90.00%	0.51	0.21 (M)
`parallel`[7]	2.1.0	XLnet	2,000	90.02%	90.16%	0.61	0.16 (S)
`py_func`	2.1.0	RetinaNet[9]	5,000	0.24	0.24	0.35	0.30 (M)

1: `soft` is `softmax_cross_entropy_with_logits`
2: `sparse` is `sparse_to_dense`
3: `exp` is `experimental_run_functions_eagerly`
4: `max_` is `max_pooling2d`
5: `batch` is `batch_normalization`
6: `map_` is `map_and_batch`
7: `parallel` is `parallel_interleave`
8: Trans is Transformer and uses **BLEU** score (Papineni, Roukos, Ward, & Zhu, 2002) to evaluate model *accuracy*.
9: RetinaNet uses Average Precision at **IoU**=.50:.05:.95(Lin et al., 2021) to evaluate model *accuracy*.

each deprecated API. Column **Train** is the training steps for each model. Columns **Acc (B)** and **Acc (A)** display average model accuracy before and after deprecated API updates, respectively. In columns ***p*-val** and ***Cohen'D***, we measure the difference between the model accuracy before and after deprecated API updates by *p*-value and *Cohen'D* respectively. In column ***Cohen'D***, S denotes a small effect, and M denotes a medium effect. Notably, there is a speacial case where the deprecated API `to_int32()` is invoked in both the `Mnist` and `Transformer` models. The *p*-value of model accuracy for `Mnist` and `Transformer` are 0.59 and 0.77, accordingly, which are significantly greater than the threshold (0.05), indicating that there is no statistically significant difference in model accuracy before and after deprecated API updates. According to table 4.10, `max_pooling2d()` in `ResNet` model has the highest *p*-value (0.86). The smallest *p*-value (0.14) is found in `dense()`

in `ResNet`, however it is still greater than the threshold (0.05), i.e, none of these deprecated APIs has *p*-value lower than the threshold (0.05). For seven deprecated APIs, *Cohen's* D depicts small effects, indicating that the model accuracy difference is insignificant. Six deprecated APIs have a medium effect of *Cohen's D*, but their *p*-value is less than the threshold (0.05), therefore there is no significant difference in model accuracy.

In this study, we trained and tested each deep learning model 40 times, 20 times with deprecated API, and 20 times with replaced APIs. However, *p*-value is influenced by sample size, while if the sample size is big enough, the *p*-value will always be smaller than the threshold (0.05). A 20 size sample may not be big enough for *p*-value. A much larger sample is not achievable due to the limited computation ability in our study. Therefore, research in future work may need to find other ways to evaluate the deep learning model accuracies.

In summary, deprecated APIs are used in deep learning projects, but some (5 out of 17) are not directly related to model computation. In our experiment, accuracy difference between deprecated API and their replacements does not appear to be significant (12 out of 17).

RQ4 Finding

Based on our 13 tests involving 12 deprecated APIs, none of the deprecated API in our experiment has p-value lower than the threshold (0.05). Cohen's D depicts a medium effect for 6 tests and a small effect for 7 tests. The result implies that accuracy changes do not incentivize the users of these APIs to migrate to the replacement APIs.

Chapter 5

Threats to Validity

In this section, we discuss the threats to the validity of our research.

External validity Threats to external validity are concerned with the extent to which we can generalize our results. In order to answer RQ3, we examined models found at *Models* with different TensorFlow versions. This TensorFlow official repository is well-maintained and does not contain numerous deprecated APIs for RQ4 analysis. In the future, we will gather more deprecated APIs for analysis from more projects. In RQ1, we only studied APIs that have `@deprecated` decorator. Our results may not generalize to general API changes.

Construct validity Threats to construct validity are concerned with the validity of our conclusions within the constraints of the dataset we used. In RQ2, we identified six rationales why APIs become deprecated. However, manual analysis is used to accomplish this identification, which may result in some bias. To reduce result bias in our experiment, we used pair review and discussed any disagreements with a third reviewer. In RQ4, we evaluated the model accuracy difference. To reduce confounding factors, we train and test models 20 times with deprecated API and repeat the training and testing a further 20 times after updating these deprecated APIs.

Internal validity Threats to internal validity are concerned with how our experiments were designed. In RQ4, we trained and tested the deep learning model using its default settings, but we reduced the number of training steps from the documentation. It is probable that the training procedures are not adequate to produce accurate results. However, these reduced training steps are appropriate because our study does not focus on data prediction. In addition, we trained and tested

each deep learning model 40 times because of the limited computation resources. This sample size may not be big enough for p-value. Increasing the sample size to several hundred may not be practical. Therefore, future research may need to find another method to evaluate model accuracy before and after deprecated API updating.

Chapter 6

Related Works

In prior research, many studies have been conducted on API evolution and its impacts on projects. We discuss some related work in this section.

API Deprecation

Deprecated APIs often follow a *deprecated-replace-remove* cycle, in which APIs are first marked as deprecated, and then replacement messages are supplied to developers to aid with code migration. Finally, these deprecated APIs are removed after a period of development time. However, API deprecation is not always appropriately addressed. Wang, Li, Liu, and Cai (2020) present an exploratory study of deprecated python APIs and discover that Python library contributors often ignore the recommended package and implement ad-hoc strategies to deprecate APIs. These strategies differ from library to library, while each library may use multiple strategies. Ko et al. (2014) investigate the document quality of deprecated APIs and reveal that only 61% of deprecated API documents provide alternative APIs while rationale and examples are rarely documented, leaving many deprecated API usages in client applications unresolved. Brito, Hora, Valente, and Robbes (2016) examine the frequency of deprecation messages in 661 real-world Java systems and discover that only 64% of the API are deprecated with replacement messages, implying that almost no significant effort is made to enhance deprecation messages over time. Zhou and Walker (2016) discover that many APIs are removed without prior deprecation or are afterward un-deprecated, and such

APIs are even resurrected with unexpected frequency. These works, on the other hand, are focused on more traditional software research areas related to Java or Android. As indicated by Zhang et al. (2020), the API evolution patterns in Python or TensorFlow largely differ. To fill this gap, our study investigates the deprecation situation in TensorFlow.

Furthermore, many researchers investigate the migration of deprecated APIs. Haryono et al. (2020) and Štrobl and Troníček (2013) propose automated approaches to update deprecated APIs in Android and Java. Lamothe and Shang (2018) presents practical experience about automated API migrating techniques based on documentation and historical code changes. They disclose that official documentation contributes the majority (75.3%) of the information to suggest API migrations. However, the automation approach in deep learning fields has received little attention and will require further research in the future. In addition, some study are conducted about the rationale behind API deprecation. Sawant et al. (2018) perform a manual analysis of 374 deprecated methods' *Javadoc* and other data sources, revealing 12 reasons why API providers deprecate a feature. Similarly, we conduct an empirical study about API deprecation reasons in TensorFlow.

Impact of API Evolution

Researchers have conducted some research on developers' reactions to deprecated APIs. Li et al. (2020) present a CDA approach for characterizing deprecated Android APIs and discover that 37.87% of APPs use deprecated APIs, with Google Play accessing deprecated APIs at a higher rate than other markets. Ko et al. (2014) investigate the quality of API documents and find that only 49% of deprecated APIs are updated if the related API documents do not provide alternative APIs. In our research, we look at the use of deprecated APIs in deep learning models, and our results indicate that 5-10% of TensorFlow deprecated APIs appear in the analyzed projects.

API evolution brings many advantages and optimizations to the framework but also introduces certain issues. Hora et al. (2018) examine 118 API changes extracted from a large-scale system and disclose that API changes can affect the entire ecosystem in terms of client systems, methods, developers. Zhang et al. (2020) find that Python framework API evolution may result in unexpected behavior in client applications and propose an approach for discovering these issues automatically.

Our study extends these efforts into deep learning to conduct an analysis of the effect of deprecated APIs on model accuracy.

Chapter 7

Conclusion

In this thesis, we conduct an empirical study on deprecated Python APIs in TensorFlow. We have investigated the evolving pattern of deprecated APIs in 20 TensorFlow releases spanning TensorFlow 1.0 to TensorFlow 2.3. In addition, we analyzed the deprecation messages for 235 deprecated APIs to determine their deprecation rationales. Moreover, we gathered 12 deep learning models and studied the deprecated API distribution in these projects. We then manually updated deprecated APIs found and evaluated accuracy differences by training and testing these models before and after deprecated APIs updating to explore the impact of deprecated APIs on deep learning models.

Our results reveal that 1) Deprecated APIs increases constantly as TensorFlow evolves while only a small portion (4.52%) of deprecated APIs are removed eventually, but their survival time is quite short (2 versions in median). Most deprecated APIs take a comparatively long period (12 versions in median) to get deprecated since they were introduced into TensorFlow. 2) The primary reasons for deprecating TensorFlow API are optimization improvements (e.g., **API Optimization** (53.77%)) for the method level and usability improvements (e.g., **Parameter Name Change** (91.67%)) for the parameter level. 3) There exist few deprecated APIs ($\sim$5-10%) in *Models* projects. 4) Among all of the deprecated APIs we analyzed, none of them has p-value lower than the threshold (0.05), and *Cohen's D* displays medium effect for 6 tests and small for 7 tests. The result implies that there is no significant accuracy difference following deprecated API updates. Therefore, accuracy changes do not incentivize the users of these APIs to migrate to the replacement APIs.

In future work, we intend to gather more deep learning projects to evaluate the deprecated APIs' impact on model performance. We also intend to extend our study to other deep learning frameworks such as PyTorch.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... others (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Bojar, O., et al. (Eds.). (2017). *Proceedings of the second conference on machine translation*. Copenhagen, Denmark: Association for Computational Linguistics. Retrieved from `http://www.aclweb.org/anthology/W17-47`

Brito, G., Hora, A., Valente, M. T., & Robbes, R. (2016). Do developers deprecate APIs with replacement messages? a large-scale analysis on java systems. In *2016 ieee 23rd international conference on software analysis, evolution, and reengineering (saner)* (Vol. 1, pp. 360–369).

Carbonnelle, P. (2021). *Pypl popularity of programming language.* `http://pypl.github.io/PYPL.html`. ([Online; accessed 15-July-2021])

Community, T. (2021). *Tensorflow case studies.* `https://www.tensorflow.org/about/case-studies`. ([Online; accessed 25-July-2021])

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dietrich, J., Jezek, K., & Brada, P. (2014). Broken promises: An empirical study into evolution problems in java programs caused by library upgrades. In *2014 software evolution week-ieee conference on software maintenance, reengineering, and reverse engineering (csmr-wcre)* (pp. 64–73).

Gu, X., Mao, Y., Han, J., Liu, J., Wu, Y., Yu, C., ... Zukoski, N. (2020). Generating representative headlines for news stories. In *Proceedings of the web conference 2020* (pp. 1773–1784).

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, *187*, 27–48.

Hargrave, M. (2021). *What is deep learning?* `https://www.investopedia.com/terms/d/deep-learning.asp`. ([Online; accessed 15-July-2021])

Haryono, S. A., Thung, F., Kang, H. J., Serrano, L., Muller, G., Lawall, J., ... Jiang, L. (2020). Automatic android deprecated-API usage update by learning from single updated example. In *Proceedings of the 28th international conference on program comprehension* (pp. 401–405).

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In *Proceedings of the ieee international conference on computer vision* (pp. 2961–2969).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173–182).

Hora, A., Robbes, R., Valente, M. T., Anquetil, N., Etien, A., & Ducasse, S. (2018). How do developers react to API evolution? a large-scale empirical study. *Software Quality Journal*, *26*(1), 161–191.

Keras. (2021). *Keras: the python deep learning API.* `https://keras.io/`. ([Online; accessed 15-July-2021])

Ko, D., Ma, K., Park, S., Kim, S., Kim, D., & Le Traon, Y. (2014). API document quality for resolving deprecated APIs. In *2014 21st asia-pacific software engineering conference* (Vol. 2, pp. 27–30).

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* (Tech. Rep.).

Kuo, W., Angelova, A., Malik, J., & Lin, T.-Y. (2019). Shapemask: Learning to segment novel objects by refining shape priors. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 9207–9216).

Lamothe, M., & Shang, W. (2018). Exploring the use of automated API migrating techniques in practice: an experience report on android. In *Proceedings of the 15th international conference on mining software repositories* (pp. 503–514).

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, *abs/1909.11942*. Retrieved from http://arxiv.org/abs/1909.11942

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Li, L., Bissyandé, T. F., Klein, J., & Le Traon, Y. (2016). Parameter values of android apis: A preliminary study on 100,000 apps. In *2016 ieee 23rd international conference on software analysis, evolution, and reengineering (saner)* (Vol. 1, p. 584-588). doi: 10.1109/SANER.2016.51

Li, L., Gao, J., Bissyandé, T. F., Ma, L., Xia, X., & Klein, J. (2020). Cda: Characterising deprecated android APIs. *Empirical Software Engineering*, 1–41.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the ieee international conference on computer vision* (pp. 2980–2988).

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740–755).

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2021). *Detection evaluation metrics used by coco.* https://cocodataset.org/#detection-eval. ([Online; accessed 15-July-2021])

Lopez, M. M., & Kalita, J. (2017). Deep learning applied to nlp. *arXiv preprint arXiv:1703.03091*.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June). Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies* (pp. 142–150). Portland, Oregon, USA: Association for Computational Linguistics. Retrieved from http://www.aclweb.org/anthology/P11-1015

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the association for computational linguistics* (pp. 311–318).

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... Fei-Fei, L. (2015). ImageNet

Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211-252. doi: 10.1007/s11263-015-0816-y

Sawant, A. A., Huang, G., Vilen, G., Stojkovski, S., & Bacchelli, A. (2018). Why are features deprecated? an investigation into the motivation behind deprecation. In *2018 ieee international conference on software maintenance and evolution (icsme)* (pp. 13–24).

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Štrobl, R., & Troníček, Z. (2013). Migration from deprecated API in java. In *Proceedings of the 2013 companion publication for conference on systems, programming, & applications: software for humanity* (pp. 85–86).

Sullivan, G. M., & Feinn, R. (2012). Using effect size—or why the p value is not enough. *Journal of graduate medical education*, *4*(3), 279–282.

TensorFlow. (2021). *Tensorflow official website.* `https://www.tensorflow.org`. ([Online; accessed 25-July-2021])

Thiese, M. S., Ronna, B., & Ott, U. (2016). P value interpretations and considerations. *Journal of thoracic disease*, *8*(9), E928.

Vallat, R. (2018). Pingouin: statistics in python. *Journal of Open Source Software*, *3*(31), 1026.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Wang, J., Li, L., Liu, K., & Cai, H. (2020). Exploring how deprecated python library APIs are (not) handled. In *Proceedings of the 28th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering* (pp. 233–244).

Wei, L., Liu, Y., & Cheung, S.-C. (2016). Taming android fragmentation: Characterizing and detecting compatibility issues for android apps. In *Proceedings of the 31st ieee/acm international conference on automated software engineering* (pp. 226–237).

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, *32*.

Yim, K. H., Nahm, F. S., Han, K. A., & Park, S. Y. (2010). Analysis of statistical methods and errors in the articles published in the korean journal of pain. *The Korean journal of pain*, *23*(1), 35.

Yu, H., Chen, C., Du, X., Li, Y., Rashwan, A., Hou, L., ... Li, J. (2020). *TensorFlow Model Garden.* `https://github.com/tensorflow/models`.

Zhang, Z., Yang, Y., Xia, X., Lo, D., Ren, X., & Grundy, J. (2021). Unveiling the mystery of api evolution in deep learning frameworks: A case study of tensorflow 2. In *2021 ieee/acm 43rd international conference on software engineering: Software engineering in practice (icse-seip)* (p. 238-247). doi: 10.1109/ICSE-SEIP52600.2021.00033

Zhang, Z., Zhu, H., Wen, M., Tao, Y., Liu, Y., & Xiong, Y. (2020). How do python framework APIs evolve? an exploratory study. In *2020 ieee 27th international conference on software analysis, evolution and reengineering (saner)* (pp. 81–92).

Zhou, J., & Walker, R. J. (2016). API deprecation: a retrospective analysis and detection method for code examples on the web. In *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering* (pp. 266–277).