# AIADA: Accuracy Impact Assessment of Deprecated Python API Usages on Deep Learning Models

Haochen Zou*
Department of Computer Science and Software Engineering, Concordia University, Montreal H3G 1M8, Canada.

* Corresponding author. Tel.: +1 4388783856; email: haochen.zou@mail.concordia.ca
Manuscript submitted July 25, 2022; Accepted November 20, 2022.

doi: 10.17706/jsw.17.269-281

**Abstract:** TensorFlow is an end-to-end open-source machine learning platform including various tools, libraries, and community resources. It supports users to use many mainstream programming languages including Python. TensorFlow contains multiple abstraction layers, with APIs play significant roles in every layers. In the version iteration of TensorFlow platform development, with the release of new TensorFlow versions, because of functionality evolution, or security and performance-related changes, some APIs eventually become unnecessary. These issues cause APIs to deprecate and influence the accuracy of deep learning models results. Prior studies have investigated API evolution and its potential impact on projects. However, their studies mainly focus on API evolution instead of API deprecation, and they do not find out how the evolution affects results of deep learning models in TensorFlow. Therefore, we present a research-based prototype tool called AIADA and apply it to different revisions of the TensorFlow platform projects code for characterizing deprecated APIs. Based on the data mined by AIADA, we develop a quantitative assessment of deprecated Python APIs usages on deep learning models accuracy. We first count the amount of TensorFlow Python APIs that are deprecated, finding out that with the development of TensorFlow version, the number of deprecated APIs increases constantly. Second, we discuss the reason behind TensorFlow Python APIs become deprecated, discover that name change, weed out, and compatibility issue lead to the main cause of deprecation. Finally, we construct a deep learning project as the comparative experiment. After comparing the results between deep learning model with TensorFlow deprecated APIs and without deprecated APIs, we conclude that using deprecated APIs will cause a 10% loss on efficiency and accuracy of deep learning model.

**Key words:** Machine learning, deep learning, deprecated API, TensorFlow.

## 1. Introduction

Application Programming Interfaces (APIs) are sets of defined functionalities provided by a programming library or framework [1]. APIs promote the reuse of existing software systems. By integrating APIs in a codebase, developers can save development time and effort by utilizing a well-tested system.

TensorFlow is an end-to-end open-source machine learning platform which contains an extensive and elastic ecosystem, including various tools, libraries, and community resources [2]. It can assist researchers to promote the development of advanced machine learning technology and enable developers to build and deploy applications supported by machine learning [3]. TensorFlow manages several abstraction layers: the heigh-level TensorFlow APIs, the mid-level TensorFlow APIs, the low-level TensorFlow APIs, the TensorFlow

kernel APIs, and the hardware layer [4]. APIs play considerable characters among multiple abstraction layers. However, with the development of new TensorFlow versions, some APIs cannot fit the requirement of the framework which results in the deprecation of APIs. To enable a graceful adaptation of developers to framework changes, API deprecations are implemented following the deprecate-replace-remove cycle [5]. In this scheme, APIs that will no longer be maintained in the framework are first flagged as deprecated, through the @deprecation Python annotation. Subsequently, the code of deprecated APIs is updated with replacement messages which are meant to help developers refactor their applications to migrate from deprecated APIs to their replacements [6]. Finally, after some reasonable time, deprecated APIs are eventually removed to clean the framework and thereby reducing the maintenance burden on the framework codebase [7].

Deep learning is a research direction in the field of machine learning. It is introduced into machine learning to make it closer to artificial intelligence [8]. Deep learning is a complex machine learning algorithm, which enables the machine imitate human activities such as audio-visual and thinking to solve the problem of pattern recognition and make progress in artificial intelligence-related technologies [9]. Researchers utilize the TensorFlow platform to conduct deep learning studies such as process datasets and create patterns in decision making. Whereas the TensorFlow Python APIs evolutions have a potential impact on projects and results. Numerous investigations have examined the influence. Zhang et al. analyze APIs evolution in deep learning frameworks on TensorFlow 2 and reveal reasons for API changes [10]. However, their study mainly focuses on the APIs evolution instead of the APIs abandonment. Moreover, some researchers have discovered that the evolution and deprecation of APIs might result in incompatibility issues. Li et al. characterize deprecated APIs in the Androids ecosystem and identified three bugs related to deprecated APIs [11]. Anand et al. indicate an investigation into the motivation behind APIs deprecation and define a taxonomy of 12 high-level reasons [12]. Ferdian et al develop a study on automated deprecated API usage updates for Android applications [13]. However, few pieces of research focus on deprecated APIs in the TensorFlow platform and the impact of deprecated APIs on deep learning models. Recent studies relevant to the APIs in the TensorFlow platform and their impacts of them on deep learning models have been proposed. Raschka et al. provide an in-depth look into the Python machine learning world and explores key topics to identify some of the core hardware and software paradigms that support it. They cover a wide range of APIs and concepts, gathering them together for an overall comparison [14]. However, their work does not address the impact of the deprecated APIs in Python machine learning on machine learning, data science, and scientific computing. Although researchers have conducted empirical studies to understand deep learning bugs, these studies focus on bugs of their applications, and the nature of bugs inside a deep library is still largely unknown. To deepen the understanding of such bugs, Jia et al. analyze 202 bugs inside TensorFlow [15]. We will further introduce research on analyzing versioning and internal deprecated APIs of the TensorFlow platform to complement empirical research on deep learning errors. Therefore, to characterize deprecated deep learning TensorFlow Python APIs, in this paper, we first design a prototype tool called AIADA. Then we apply AIADA to different revisions of the TensorFlow platform projects code and compare the obtained results to understand the evolution of deprecated TensorFlow Python APIs. The overall goal of this research is to draw insights into (1) the quantity and features of deprecated APIs, (2) the reason for APIs becoming abandoned, and (3) the potential impact of deprecated APIs on deep learning models.

In this work, we present an exploratory study on the deprecation of TensorFlow Python APIs. This study builds on a systematic source code mining of the TensorFlow framework, which covers 23 TensorFlow releases spanning versions 1.0 to 2.6. This study also involves analyzing 22 deep learning models provided by the official TensorFlow GitHub organization and maintained by Intel and NIVIDA which cover the field of computer vision, object detection, segmentation, natural language processing, and recommendation systems. We first design and implement a prototype tool called AIADA. AIADA is based on the Abstract Syntax Tree

(AST) of source code. Then we apply AIADA to 23 different revisions of the TensorFlow framework code and 22 deep learning models to conduct the statistical quantity of deprecation APIs. After that, we explore the deprecation messages extracted by AIADA AST to label deprecation features. Finally, we construct a deep learning project as a comparative experiment to assess the accuracy impact of deprecated API usages on deep learning models.

## 2. Materials and Methods

### 2.1. Deprecated APIs Quantity Statistics Based on AST

One of the core problems in collecting deprecated APIs from the TensorFlow Python deep learning library is that each of these deep learning libraries has a different way of documenting their API deprecations and different software engineers have diverse code styles [16]. To address this problem, we invest in an abstract syntax tree for deprecated API data collection. The abstract syntax tree (AST) represents the abstract syntax structure of the program source code that is expected to be analyzed as a tree structure [17]. Each node in the tree structure represents a structure type in the code fragment of the program to be analyzed, the structure of the program source code and the execution process of the source code are reflected by establishing the relationship between the parent node and brother node. The abstract syntax tree is used to represent the structural information between program codes to realize the analysis of source code [18]. The abstract syntax tree does not represent every detail of the real program syntax, but only retains the key syntax content of the program source code with analysis to realize the abstract representation of the program source code [19]. After the syntax structure of program code is abstractly identified and displayed as a tree structure, when compared with simple code text analysis, it is clear to acquire its code structure and syntax rules to find out deprecated APIs based on semantic information.

We focused on looking for the APIs Python annotations containing the word “deprecate” and “replace”, which indicates API deprecation or replacement. Each input Python file is first transformed into its abstract syntax tree by AIADA using the abstract syntax tree module provided by Python. From the abstract syntax tree, all the public classes and methods are traversed. AIADA detects and locates the usage of the deprecated APIs based on the deprecated APIs' names and annotations syntax signature. Finally, methods and classes with deprecated APIs will be flagged by @deprecated and added to a statistics database. The architecture and pipeline of AIADA collecting deprecated APIs are given in Figure 1.

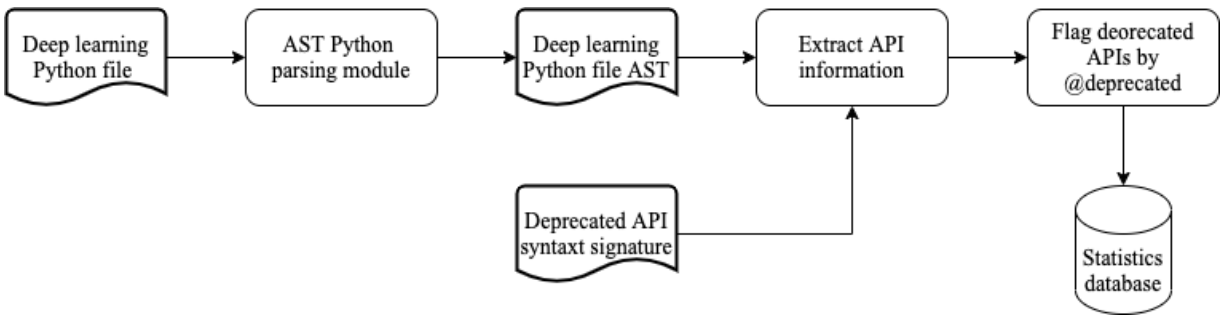


Fig. 1. AIADA architecture and pipeline.

### 2.2. Comparative Experiment Design

In this paper, we design a target tracking program based on deep learning models in the TensorFlow platform for the experiment. The experimental framework constructed in this paper is based on the TensorFlow Python library. TensorFlow utilizes graphs to represent computing tasks [20]. The nodes in the graph are defined as *op* (abbreviation for operation). A node obtains zero or more tensors, and each tensor is

a typed multidimensional array. TensorFlow program can be divided into two stages: construction graph and execution graph [21]. A construction graph is a calculation diagram that TensorFlow needs to build before performing calculation operations. The calculation diagram is composed of nodes and corresponding operations between nodes. At the same time, the required constants and variables need to be defined. The execution graph can only be run after the construction graph is created. The first step of the execution graph is to create a session object, called the start of the graph, then initialize the variables, perform the corresponding operations between nodes, update the variable values, and finally save the final data. The framework structure of the deep learning model for image recognition and tracking designed in this paper based on TensorFlow is shown in Fig. 2 [22].

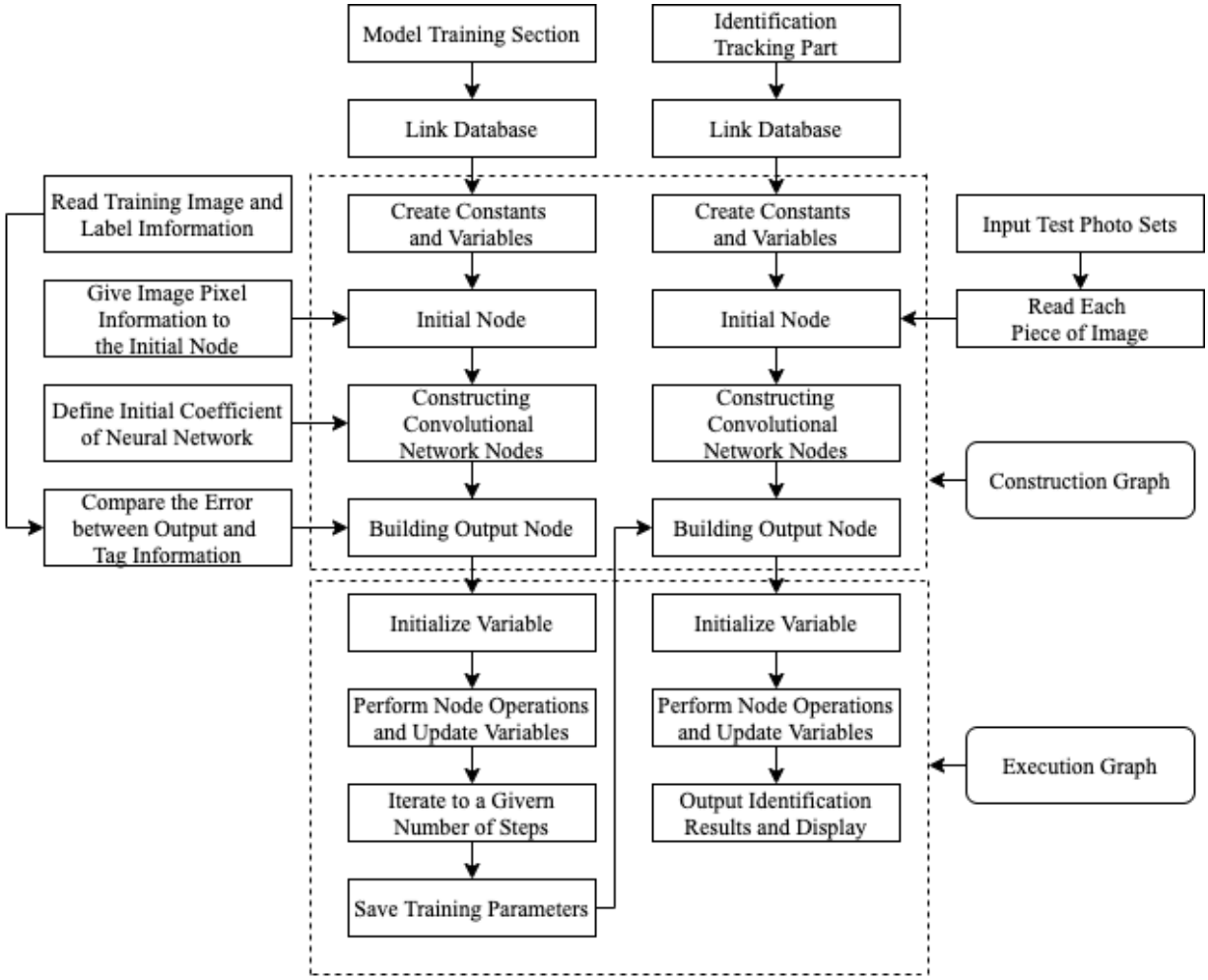


Fig. 2. The framework structure of the deep learning model for image recognition and tracking.

The experiment is divided into two parts: the model training part and the recognition tracking part. The node construction and inter-node operation of the two parts are the same. The main difference lies in the initial assignment of weight variables. The initial weights of the training part of the model are given artificially, and the weights of each layer of the frame structure are adjusted by training samples to make the errors between the actual output and the ideal output smaller. The weights of the recognition tracking part are directly used to obtain the weights of the model training part, the test video information is mapped through each layer network, and the final output is our recognition result.

The data utilized in this experiment is the basketball photo set downloaded from the VOT2020 standard dataset. The pixel size of images is 800*480. 100 images are used as training samples and another 100 images

as test samples. The identified targets are marked with white rectangular boxes. For the training sample set, each image is processed as follows: mark the pixel position *x* and *y* of the target center point, as well as the length *h* and width *w* of the rectangular frame, and then normalize the data. Data normalization refers to putting each attribute of the input feature into a unified interval so that each attribute has the same contribution to classification.

## 2.3. Construction of Deep Learning Model

At the beginning of the experiment, the relevant function library should be introduced. A construction graph is a key part of deep learning model design [23]. Its main work is to complete the creation of nodes and define the operation mode between nodes. Since the establishment and operation of nodes rely on tensors, the node connection of the model is usually represented by the connection of tensors [24]. A set of images is usually represented as a four-dimensional floating-point number group, and the four dimensions are Batch, Height, Width, and Channels.

In this experiment, the learning method is the Adam algorithm, namely the adaptive moment estimation method. Adam algorithm dynamically adjusts the learning rate of each parameter according to the first and second moment estimation of the gradient of each parameter by the loss function [25]. The Adam algorithm is also based on the gradient descent method, but the learning steps of each iteration parameter have a certain range, and a large gradient will not lead to a large learning step, so the parameter value is stable [26].

At the beginning of the training, we need to initialize the gradient cumulant and the square cumulant:

$$V_{dw} = 0, V_{db} = 0, S_{dw} = 0, S_{db} = 0$$

Suppose that in the $t$ round of training, we can first calculate the parameter update:

$$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) dW$$

$$V_{db} = \beta_1 V_{db} + (1 - \beta_1) db$$

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) dW^2$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2) db^2$$

Since the moving index average at the beginning of iteration will lead to a large difference from the initial value, we need to correct the deviation of several values obtained above.

$$V_{dw}^c = \frac{V_{dw}}{1 - \beta_1^t}$$

$$V_{db}^c = \frac{V_{db}}{1 - \beta_1^t}$$

$$S_{dw}^c = \frac{S_{dw}}{1 - \beta_2^t}$$

$$S_{db}^c = \frac{S_{db}}{1 - \beta_2^t}$$

Through the above formula, $V_{dw}$, $V_{db}$, $S_{dw}$, and $S_{db}$ respectively are the gradient momentum accumulated by the loss function during the first $t-1$ iteration. we can obtain the modified value of the

parameter gradient cumulant in the $t$ round of iteration, so that the weight and bias can be updated.

$$W = W - \alpha \frac{V_{dw}^c}{\sqrt{S_{dw}^c} + \epsilon}$$

$$b = b - \alpha \frac{V_{db}^c}{\sqrt{S_{dw}^c} + \epsilon}$$

In the algorithm, we define $\beta_1 = 0.9$, $\beta_2 = 0.99$. $\epsilon$ is a smooth top and we define $\epsilon = 10^{-8}$. Learning rate $\alpha$ requires fine-tuning during training.

## 3. Results and Discussion

### 3.1. Quantity and Feature of Deprecated APIs

From the results of AIADA, we first detect the quantity character of deprecated Python APIs. We discover that with the development of TensorFlow versions, both APIs and abandoned APIs of TensorFlow increase jointly as shown in Table 1.

Table 1. Quantity of Deprecated APIs among TensorFlow Versions Evolvement

| TensorFlow Version | Python Version | Release Date | APIs | Deprecated APIs |
|---|---|---|---|---|
| TensorFlow 2.6.0 | 3.6-3.9 | 2021.08.09 | 6638 | 283 |
| TensorFlow 2.5.0 | 3.6-3.9 | 2021.06.03 | 6605 | 257 |
| TensorFlow 2.4.0 | 3.5-3.8 | 2020.11.07 | 6537 | 219 |
| TensorFlow 2.3.0 | 3.5-3.8 | 2020.06.10 | 6493 | 192 |
| TensorFlow 2.2.0 | 3.5-3.8 | 2020.05.08 | 6381 | 168 |
| TensorFlow 2.1.0 | 2.7, 3.5-3.7 | 2019.12.21 | 6005 | 149 |
| TensorFlow 2.0.0 | 2.7, 3.3-3.7 | 2019.09.19 | 5722 | 142 |
| TensorFlow 1.15.0 | 2.7, 3.3-3.7 | 2019.09.18 | 5628 | 142 |
| TensorFlow 1.14.0 | 2.7, 3.3-3.7 | 2019.06.08 | 5267 | 133 |
| TensorFlow 1.13.0 | 2.7, 3.3-3.7 | 2019.03.15 | 4431 | 107 |
| TensorFlow 1.12.0 | 2.7, 3.3-3.6 | 2018.10.08 | 3809 | 54 |
| TensorFlow 1.11.0 | 2.7, 3.3-3.6 | 2018.09.13 | 3512 | 45 |
| TensorFlow 1.10.0 | 2.7, 3.3-3.6 | 2018.08.07 | 3362 | 38 |
| TensorFlow 1.9.0 | 2.7, 3.3-3.6 | 2018.07.11 | 3265 | 32 |
| TensorFlow 1.8.0 | 2.7, 3.3-3.6 | 2018.04.19 | 3189 | 27 |
| TensorFlow 1.7.0 | 2.7, 3.3-3.6 | 2018.03.29 | 3025 | 25 |
| TensorFlow 1.6.0 | 2.7, 3.3-3.6 | 2018.03.27 | 2983 | 25 |
| TensorFlow 1.5.0 | 2.7, 3.3-3.6 | 2018.01.18 | 2806 | 20 |
| TensorFlow 1.4.0 | 2.7, 3.3-3.6 | 2017.11.01 | 2630 | 18 |
| TensorFlow 1.3.0 | 2.7, 3.3-3.6 | 2017.08.22 | 2011 | 16 |
| TensorFlow 1.2.0 | 2.7, 3.3-3.6 | 2017.06.14 | 1879 | 15 |
| TensorFlow 1.1.0 | 2.7, 3.3-3.6 | 2017.04.21 | 1662 | 12 |
| TensorFlow 1.0.0 | 2.7, 3.3-3.6 | 2017.02.11 | 1508 | 12 |

We explore the result of AIADA. For the growth trend of the total number of Python APIs in the TensorFlow platform, we indicate that APIs in the TensorFlow platform increased significantly between the release date 2017.08.22 and 2017.11.01 from TensorFlow version 1.3.0 to 1.4.0. The number of APIs also grow drastically between the release date 2018.10.08 and 2019.09.18 from TensorFlow version 1.12.0 to 1.15.0. For the rest of the time, the number of APIs grows relatively slowly. The growth trend of a total number of Python APIs in the TensorFlow platform is displayed in Fig. 3.

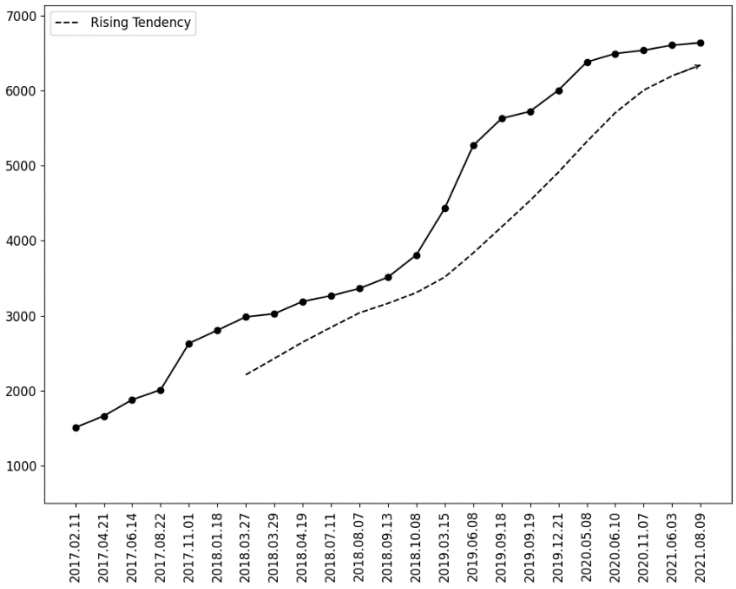


Fig. 3. The framework structure of the deep learning model for image recognition and tracking.

To find the characters of the growing trend of the number of deprecated Python APIs in the TensorFlow platform, we figure out that deprecated APIs in the TensorFlow platform increased significantly between the release date 2018.10.08 and 2019.06.08 from TensorFlow version 1.12.0 to 1.14.0. The number of APIs also see a drastic growth after the release date 2019.12.21 till now from TensorFlow version 2.1.0. While during the rest of the release time, the number of deprecated APIs raise relatively slowly. The growth trend of the number of deprecated Python APIs in the TensorFlow platform is displayed in Fig. 4.

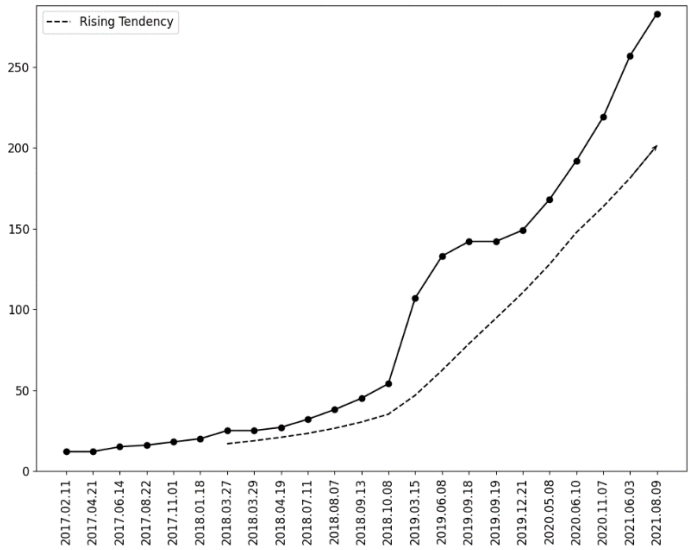


Fig. 4. The growth trend of the number of deprecated Python APIs in TensorFlow platform.

Fig. 5 presents the proportion of deprecated APIs in all TensorFlow Python APIs among different release times. The proportion is comparatively steady from TensorFlow version 1.0.0 to 1.8.0 between release time 2017.02.11 and 2018.04.19 which is 0.8%. The proportion is also relatively stable from TensorFlow version 1.13.0 to 2.1.0 between release time 2019.03.15 and 2019.12.21 which is 2.5%. During the rest of the release time, with the evolvement of TensorFlow versions, the proportion of deprecated APIs in all TensorFlow Python APIs increased dramatically.

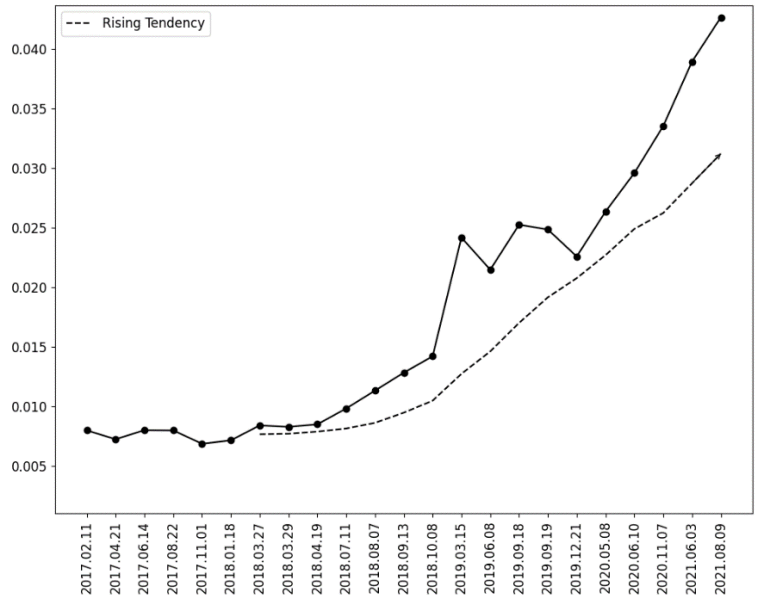


Fig. 5. The proportion of deprecated APIs in all TensorFlow Python APIs.

The conclusion indicates that with the development of TensorFlow versions, the number of deprecated APIs and the proportion of deprecated APIs in all TensorFlow Python APIs increase jointly. Since the proportion of deprecated APIs in all TensorFlow APIs has increased significantly after release date 2019.12.21 till now, we can indicate from the results of AIADA that developers of TensorFlow do not respect the deprecated-replace-remove cycle and deprecated APIs still exist in the TensorFlow platform library and deep learning models.

## 3.2. The Reason for TensorFlow APIs Becoming Deprecated

We aim to detect the reason for TensorFlow APIs becoming abandoned. After TensorFlow 2.0, nearly every TensorFlow evolvement will lead to deprecated APIs incensement. To discover the reason, we first explore the deprecation annotation message extracted by AST in AIADA. If there is no deprecated annotation or the content of annotation is inaccurate, we will check the official document about the APIs. We will also compare the source of deprecated APIs with their substitution APIs if the substitution APIs are mentioned in the official TensorFlow API documentation and the version iteration description.

The first reason is the name change. As TensorFlow evolves, some APIs' names may not reflect the current functionality [27]. Deprecated APIs will be given new names by developers while the underlying source code and functionality remain the same [28]. Some API parameters' names will be also updated during evolution to be more intuitive and enhance user-friendliness [29]. These name changes of API parameters will cause the abandonment of APIs.

The second reason is to weed out. With the development of versions, some features will be removed from the TensorFlow library which will lead to APIs with these features deprecated [30]. Some parameters could become redundant and be abandoned due to the changes in API logic, which lead to API deprecation. Some APIs are no longer able to meet the increasing requirements of the TensorFlow platform. As a result, developers introduce newer APIs to replace the aged ones to improve performance and organization [31].

The third reason is the compatibility issue. When new features introduce to the TensorFlow platform, some APIs will face compatibility problems. These APIs will be prohibited by developers [32].

Among the reasons listed above, in 283 deprecated APIs from the latest release date version of TensorFlow, weed out is the main reason for API being abandoned, taking up 54.77% which is 155 out of 283. The name

change is the second primary result that caused the API deprecation, taking up 32.5% which is 92 out of 283. Compatibility issue is the least reason for API becoming deprecated, taking up 12.72% which is 36 out of 283.

When comparing the result with java, Sawant et al. analyzed 4 Java frameworks and found that functional defect is the leading cause for API being deprecated in Java which is similar to the result we proposed. From the conclusion of Sawant et al., in Java, renaming of features only takes a small part of deprecated APIs while among deprecated TensorFlow Python APIs, it is the second major cause.

### 3.3. The Influence of Deprecated APIs on the Recognition Effect of Deep Learning

The identification error is the quadratic root difference between the target center point pixel value obtained by the test and the actual target center point pixel value, namely, the center position error, which is defined as:

$$\varepsilon = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

$\boldsymbol{x_1}$ and $\boldsymbol{y_1}$are the positions of the identified target center. $\boldsymbol{x_0}$ and $\boldsymbol{y_0}$ are the positions of the marking center. $\boldsymbol{x_0}$ and $\boldsymbol{y_0}$ are the normalized value, the identification error is also the normalized error.

The unit of time is $\boldsymbol{s}$. The training time of each step of the model was 9.527$\boldsymbol{s}$. The recognition time of each image is 0.2285$\boldsymbol{s}$. As the number of training steps increases, the curve of recognition error is shown in Figure 6 as follows. As can be seen from Fig. 6, with the increase in the number of training steps, the recognition accuracy is also improving. However, when the number of training steps reaches 1200, the recognition accuracy hardly changes with the increase of training steps. This indicates that the weight obtained by deep learning training has reached stability.

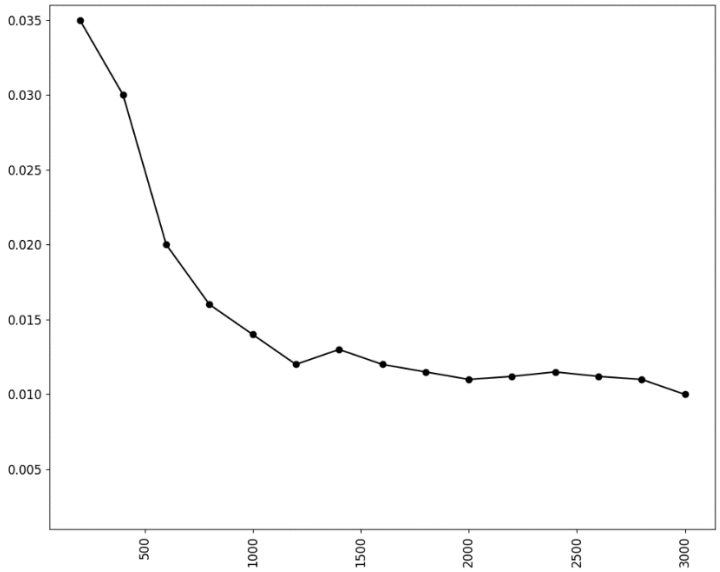


Fig. 6. The curve of recognition error before applied deprecated APIs.

Then we replaced all APIs in the program with deprecated APIs according to AIADA detection results and repeated the above experimental steps. The training time of each step of the model was 10.457$s$. The recognition of each image is 0.2512 $s$. Both the training time of each step of the model and the recognition time of each image increase 10%. As the number of training steps increases, the curve of recognition error is shown in Fig. 7 as follows. Figure 7 presents that with increase of the number of training steps, the recognition accuracy is also improving, and the recognition accuracy tends to be stable and hardly changes after step 1200. However, the overall accuracy of recognition after replacing APIs by deprecated APIs decreased 10%

which indicates that deprecated TensorFlow Python API usages have almost 10% adverse impact on deep learning models.

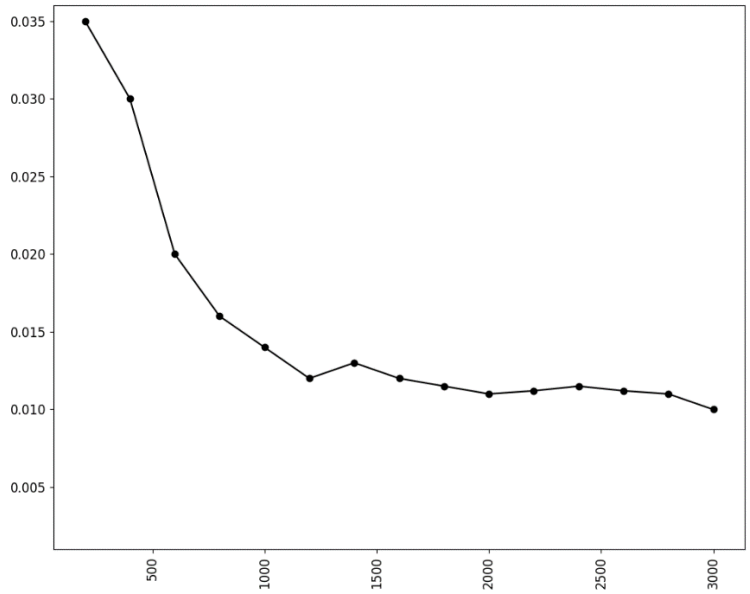


Fig. 7. The curve of recognition error after applied deprecated APIs.

## 4. Conclusions

Understanding the accuracy impact of deprecated python API usages on deep learning models helps developers better design their deep learning program and improve operating efficiency. This paper proposed a research-based prototype tool called AIADA to detect deprecated Python APIs and apply it to different revisions and models of the TensorFlow platform projects code. We discover that with the development of TensorFlow versions, the number of deprecated APIs and the proportion of deprecated APIs in all TensorFlow Python APIs increase jointly. Then we analyze the result from AIADA to discover the reason for APIs being abandoned in TensorFlow. There are three main reasons for APIs becoming deprecated: name change, weed out, and compatibility issues. Finally, we design a deep learning program for experiments and compare the results of recognition errors from the latest updated APIs and deprecated APIs. Results of the implementation and the experiment demonstrate that deprecated APIs in deep learning programs will lead to a 10% loss in accuracy and efficiency contradistinguish from the program with the latest API usages.

### Conflict of Interest

The authors declare no conflict of interest

### Author Contributions

Haochen Zou is the single author. Therefore, he did everything such as constructing a model, analyzing the data, and writing the paper.

### References

[1] Sawant, A. A., Huang, G., Vilen, G., Stojkovski, S., & Bacchelli, A. (2018, September). Why are features deprecated? an investigation into the motivation behind deprecation. *Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution* (pp. 13-24).

[2] Lee, T., Singh, V. P., & Cho, K. H. (2021). Tensorflow and keras programming for deep learning. *Deep Learning for Hydrometeorology and Environmental Science*.

[3] Kraska, T., Talwalkar, A., Duchi, J. C., Griffith, R., Franklin, M. J., & Jordan, M. I. (2013, January). MLbase: A distributed machine-learning system.

[4] Blaiech, A. G., Khalifa, K. B., Valderrama, C., Fernandes, M. A., & Bedoui, M. H. (2019). A survey and taxonomy of FPGA-based deep learning accelerators. *Journal of Systems Architecture*, *98*, 331-345.

[5] Li, L., Gao, J., Bissyandé, T. F., Ma, L., Xia, X., & Klein, J. (2018, May). Characterising deprecated android apis. *Proceedings of the 15th International Conference on Mining Software Repositories* (pp. 254-264).

[6] Xing, Z., & Stroulia, E. (2007). API-evolution support with Diff-CatchUp. *IEEE Transactions on Software Engineering*, *33(12)*, 818-836.

[7] Wang, J., Li, L., Liu, K., & Cai, H. (2020, November). Exploring how deprecated python library apis are (not) handled. *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 233-244).

[8] Nguyen, G., Dlugolinsky, S., Bobák, M., Tran, V., Lopez Garcia, A., Heredia, I., ... & Hluchý, L. (2019). Machine learning and deep learning frameworks and libraries for large-scale data mining: A survey. *Artificial Intelligence Review*, *52(1)*, 77-124.

[9] Dargan, S., Kumar, M., Ayyagari, M. R., & Kumar, G. (2020). A survey of deep learning and its applications: a new paradigm to machine learning. *Archives of Computational Methods in Engineering*, *27*(*4*), 1071-1092.

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

[11] Li, L., Gao, J., Bissyandé, T. F., Ma, L., Xia, X., & Klein, J. (2020). Cda: Characterising deprecated android apis. *Empirical Software Engineering*, *25*(*3*), 2058-2098.

[12] Sawant, A. A., Robbes, R., & Bacchelli, A. (2019). To react, or not to react: Patterns of reaction to API deprecation. *Empirical Software Engineering*, *24*(6), 3824-3870.

[13] Thung, F., Haryono, S. A., Serrano, L., Muller, G., Lawall, J., Lo, D., & Jiang, L. (2020, February). Automated deprecated-api usage update for android apps: How far are we?. *Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 602-611).

[14] Raschka, S., Patterson, J., & Nolet, C. (2020). Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, *11(4)*, 193.

[15] Jia, L., Zhong, H., Wang, X., Huang, L., & Lu, X. (2021). The symptoms, causes, and repairs of bugs inside a deep learning library. *Journal of Systems and Software*, *177*, 110935.

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

[17] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., & Liu, X. (2019, May). A novel neural source code representation based on abstract syntax tree. *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)* (pp. 783-794).

[18] Fang, C., Liu, Z., Shi, Y., Huang, J., & Shi, Q. (2020, July). Functional code clone detection with syntax and semantics fusion learning. *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 516-527).

[19] Savić, M., Rakić, G., Budimac, Z., & Ivanović, M. (2014). A language-independent approach to the

extraction of dependencies between source code entities. *Information and Software Technology*, *56*(*10*), 1268-1288.

[20] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). {TensorFlow}: A system for {large-scale} machine learning. *Proceedings of the 12th USENIX symposium on operating systems design and implementation (OSDI 16)* (pp. 265-283).

[21] Zeng, Z., Gong, Q., & Zhang, J. (2019, March). CNN model design of gesture recognition based on tensorflow framework. *Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)* (pp. 1062-1067).

[22] Liu, B., Wang, S. Z., Xie, Z. X., Zhao, J., & Li, M. (2019). Ship recognition and tracking system for intelligent ship based on deep learning framework. *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation*, *13*.

[23] Abadi, M., Isard, M., & Murray, D. G. (2017, June). A computational model for TensorFlow: an introduction. *Proceedings of the Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (pp. 1-7).

[24] Rasmussen, D. (2019). NengoDL: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics*, *17*(*4*), 611-628.

[25] Zhao, H., Liu, F., Zhang, H., & Liang, Z. (2019). Research on a learning rate with energy index in deep learning. *Neural Networks*, *110*, 225-231.

[26] Daniel, C., Taylor, J., & Nowozin, S. (2016, February). Learning step size controllers for robust neural network training. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*.

[27] Zhang, X., Zhang, Y., Zhong, M., Ding, D., Cao, Y., Zhang, Y., ... & Yang, M. (2020, October). Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security* (pp. 757-770).

[28] Zhang, Z., Zhu, H., Wen, M., Tao, Y., Liu, Y., & Xiong, Y. (2020, February). How do Python framework APIs evolve? an exploratory study. *Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (Saner)* (pp. 81-92). IEEE.

[29] Dig, D., & Johnson, R. (2006). How do APIs evolve? A story of refactoring. *Journal of software maintenance and evolution: Research and Practice*, *18*(*2*), 83-107.

[30] Jia, L., Zhong, H., Wang, X., Huang, L., & Lu, X. (2020, September). An empirical study on bugs inside tensorflow. *Proceedings of the International Conference on Database Systems for Advanced Applications* (pp. 604-620). Springer, Cham.

[31] Lübke, D., Zimmermann, O., Pautasso, C., Zdun, U., & Stocker, M. (2019, July). Interface evolution patterns: balancing compatibility and extensibility across service life cycles. *Proceedings of the 24th European Conference on Pattern Languages of Programs* (pp. 1-24).

[32] Zhang, R., Xiao, W., Zhang, H., Liu, Y., Lin, H., & Yang, M. (2020, October). An empirical study on program failures of deep learning jobs. *Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)* (pp. 1159-1170).

**Haochen Zou** received his B.S in the School of Information and Control Engineering from Shenyang Jianzhu University, Shenyang, China in 2019. He is a master student at Gina Cody School of Engineering and Computer Science in Concordia University, Montreal, Canada. His research interests is machine learning, natural language process and knowledge-based system.