An Empirical Study of the Dependency Networks of Deep Learning Libraries

Junxiao Han*, Shuiguang Deng*†, David Lo‡, Chen Zhi*†, Jianwei Yin*, Xin Xia§
*College of Computer Science and Technology, Zhejiang University, Hangzhou, China
†Alibaba-Zhejiang University Joint Institute of Frontier Technologies, Hangzhou, China
‡School of Information Systems, Singapore Management University, Singapore, Singapore
§Faculty of Information Technology, Monash University, Melbourne, Australia
{junxiaohan, dengsg, zjuzhichen, zjuyjw}@zju.edu.cn, davidlo@smu.edu.sg, xin.xia@monash.edu

***Abstract*—Deep Learning techniques have been prevalent in various domains, and more and more open source projects in GitHub rely on deep learning libraries to implement their algorithms. To that end, they should always keep pace with the latest versions of deep learning libraries to make the best use of deep learning libraries. Aptly managing the versions of deep learning libraries can help projects avoid crashes or security issues caused by deep learning libraries. Unfortunately, very few studies have been done on the dependency networks of deep learning libraries. In this paper, we take the first step to perform an exploratory study on the dependency networks of deep learning libraries, namely, Tensorflow, PyTorch, and Theano. We study the project purposes, application domains, dependency degrees, update behaviors and reasons as well as version distributions of deep learning projects that depend on Tensorflow, PyTorch, and Theano. Our study unveils some commonalities in various aspects (e.g., purposes, application domains, dependency degrees) of deep learning libraries and reveals some discrepancies as for the update behaviors, update reasons, and the version distributions. Our findings highlight some directions for researchers and also provide suggestions for deep learning developers and users.**

I. Introduction

An emerging branch of machine learning algorithms known as deep learning (DL) algorithms has attracted considerable attentions in both academia and industry recently [17]. Due to its high accuracy and advanced performance when handling various tasks, it has obtained enormous success in many cutting-edge domains, e.g., image processing [6], [10], [20], disease diagnosis [31], [32], natural language processing (NLP) [34], auto-driving [8], [22], speech and audio processing [21], and strategy gaming [30].

Deep learning algorithms convert the input to output by using multiple layers of transformation functions, where each layer successively learns information flows from front neural layers to the rear ones [23]. To implement deep learning algorithms, DL libraries (e.g., Tensorflow and PyTorch) are provided to help realize the demands of intelligent software, which in turn, allows practitioners and researchers use deep learning technologies better.

Due to the popularity of deep learning, there are many empirical studies in software engineering domains that look into deep learning code, e.g., detecting and locating code mistakes in DL applications [41], understanding the bugs types, root causes, impacts, and common antipatterns in buggy software [23], characterizing the internal behaviors of RNNs via quantitative analysis of RNN-based DL systems [14], and characterizing deep learning development and deployment across different frameworks and platforms [17]. Among them, very few studies have quantitatively analyzed the dependency management of deep learning ecosystems, not to mention the commonalities and discrepancies of deep learning projects that depend on different deep learning libraries.

Deep learning libraries are constantly evolving to add new features and fix bugs. To take full advantage of deep learning libraries, users should always keep up to date with the latest versions of deep learning libraries. Therefore, an exploratory study to understand the evolution of dependency management of deep learning libraries can shed light on the development and improvement of deep learning libraries, and provide practical suggestions to developers, users, and researchers.

To achieve this goal, in this paper, we analyze the project purposes, application domains, dependency degrees, update behaviors, and update reasons as well as the version distributions of deep learning projects that depend on different deep learning libraries, namely Tensorflow, PyTorch, and Theano. All three deep learning libraries are typical and widely studied. [19], [23], [41].

Consequently, some of the key findings include: 1) Tensorflow-dependent and PyTorch-dependent projects that provide replication packages of research papers (purpose = "paper experiments") have more contributors and stars; 2) the image and video processing, NLP, model theory (i.e., related to the basic deep learning model, such as model implementation, model improvement, etc.), and efficiency library (i.e., provide effective environments, libraries, packages, frameworks to accelerate the development process) applications are the most common application domains; 3) projects account for a higher proportion of direct dependencies (e.g., depend on deep learning library directly) than transitive dependencies (e.g., rely on other deep learning projects to implement algorithms) on deep learning libraries; 4) only a small percentage of projects have upgraded deep learning libraries out of various reasons, e.g., to address severity vulnerability, to be compatible with existed frameworks or libraries, etc.

In summary, we have the following main contributions:

*Shuiguang Deng is the corresponding author.

- To the best of our knowledge, we are the first to perform an empirical study of the dependency management of open source projects that depend on Tensorflow, PyTorch, and Theano.
- We provide avenues for deep learning developers, users, and researchers to have a comprehensive understanding of the purposes, applications, dependency degrees as well as dependency versions of open source projects that rely on Tensorflow, PyTorch, and Theano.
- Our analysis highlight some practical implications for deep learning developers, users, and software engineering researchers, e.g., developers and researchers should make more efforts to provide more effective automatic tools to help manage the library versions, and can also provide empirical evidence of upgrade success to foster the upgrade behaviors.

Paper Organization. The remainder of this paper is organized as follows. We introduce the research methodology in Section 2, and present the findings of research questions in Section 3. Section 4 discusses implications and threats to validity of our study and Section 5 reviews related work. Finally, Section 6 concludes this paper and gives directions to future work.

II. Methodology

To study the dependency networks of deep learning libraries, we collected the open source projects that depend on Tensorflow, PyTorch, and Theano in GitHub. The collected data will be analyzed in the rest of the paper.

A. Research Questions

Our study aims at providing answers for the following four research questions:

- **RQ1.** *What are the purposes and applications of deep learning projects that depend on Tensorflow, PyTorch, and Theano?*
  In this RQ, we want to investigate the reasons why users use deep learning libraries and which domain issues they aim to solve, so that we can provide insights into the distributions and impacts of purposes and applications.
- **RQ2.** *To what extent do deep learning projects depend on Tensorflow, PyTorch, and Theano?*
  Some deep learning projects are apt to rely on deep learning libraries to achieve their functionality directly, but there are also some deep learning projects that depend on other deep learning projects to avoid reimplementing the same functionality. This kind of inter-project dependence may increase the risk of maintainability issues and failures. For instance, if a deep learning project do not manage the deep learning library and the current version of the deep learning library has a security vulnerability, then, other deep learning projects that depend on this project will also be affected. Therefore, we aim to answer this RQ to reveal the dependence degrees of deep learning projects and compare if there exists a difference of deep learning projects that depend on Tensorflow, PyTorch, and Theano.

TABLE I
The statistics of the dataset for open source projects in GitHub that depend on Tensorflow, PyTorch, and Theano.

Library	All projects	Remained projects	Studied projects
Tensorflow-dependent	46,930	14,328	708
PyTorch-dependent	15,812	6,831	339
Theano-dependent	5,620	2,063	103

- **RQ3.** *How are the update behaviors of deep learning projects that depend on Tensorflow, PyTorch, and Theano?*
  This research question concerns library migration of deep learning projects. We aim to understand and compare the update behaviors and update reasons for deep learning projects that depend on Tensorflow, PyTorch, and Theano. Therefore, we can help deep learning library users manage their projects better and provide insights for deep learning developers to improve deep learning libraries.
- **RQ4.** *How often do deep learning projects use the latest versions of Tensorflow, PyTorch, and Theano?*
  This research question aims to study the distribution of dependency versions of deep learning libraries. By finding out the usage status of deep learning libraries, it can help improve deep learning projects better.

B. Data Collection

We extracted open source projects in GitHub that belong to the "used by" list of Tensorflow, PyTorch, and Theano via the dependency graph provided by GitHub API. As a result, we obtained 46,930, 15,812, and 5,620 projects that depend on Tensorflow, PyTorch, and Theano, respectively. All the data were collected up to December 2019.

We then removed projects that were forked, non-starred, and deleted [29] to further refine the projects in our dataset. We did not limit our projects to the most popular ones, so that we can have a comprehensive understanding of the dependency management status. Ultimately, for Tensorflow-dependent projects, 32,602 projects were dropped and 14,328 projects remained. For PyTorch-dependent projects, 8,981 projects were discarded and 6,831 projects remained. And for Theano-dependent projects, we excluded 3,557 projects and 2,063 projects remained.

Next, to deeply understand the characteristics of the collected projects, we conducted a sampling process by selecting 5% of the remained projects and stratified selection process according to the popularity. Notably, by manually checking, it is sufficient to select 5% of the remained projects to perform our study. As a result, we obtained 708, 339, and 103 projects for Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects, respectively. The extracted sample projects will be studied in the rest of the paper. The statistics of the projects can be seen in Table I.

We used the GitHub API to extract more information about the deep learning projects: full name, description, readme content, main programming language, number of stars, number of contributors, etc.

C. Classification and Labeling

To determine the purposes and applications of open source projects that depend on Tensorflow, PyTorch, and Theano, we look into the project name, description, label, readme content, and information of owner's homepage to manually categorize these extracted sample projects by the card sorting approach [23], [36], [37].

Classification. To classify the purposes of sample projects, we referred to the purposes of projects defined in Kalliamvakou et al.'s study [24] (e.g., software development, experimental, academic, etc.) and adapted on top of that to fit our dataset better.

We first used the 708 sample projects that depend on Tensorflow to manually determine its purpose categories. We divided the 708 sample projects into different collections according to their purposes. Next, the first and fourth authors discussed each collection to determine a suitable name for each collection. Consequently, we determined 5 purposes, which is shown in Table II. Specially, we use the "Other" category to represent the projects that cannot determine their purposes or projects that cannot fit any other categories.

Different projects always have various applications. We also used the 708 sample projects that depend on Tensorflow to identify their applications. We have adapted the applications for deep learning from [13], [27] (e.g., speech and audio, natural language processing, image, video, and multimodality, etc.) and added on top of that to characterize our dataset better. Two experts from natural language processing and computer vision domains supported our work and came up with the application categories jointly with the first and fourth authors. The projects were also put into application categories following a card sorting approach [23], [36], [37]. As a result, 13 application categories were derived, which is shown in Table III. Particularly, we also use the "Other" category to represent the projects that cannot determine their applications or projects that cannot fit any other categories.

Labeling. Since we have obtained the classification criteria, we used those criteria to label all other sample projects that depend on PyTorch and Theano. The first and fourth authors independently label the projects. After that, we use Fleiss Kappa [15] to understand the agreement between the two labelers. As a result, the Kappa values between two labelers of purpose categories on Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects are 0.73, 0.76, and 0.78, respectively, which all represent a substantial agreement.

Meanwhile, the Kappa values between two labelers of application categories on Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects are 0.93, 0.89, 0.99, respectively, where all reach an almost perfect agreement. We then discussed the results with ambiguous categories and reached the final decision.

D. Dependency Analysis

To determine to what extent deep learning projects depend on deep learning libraries, we check the import statements in deep learning programs to identify whether those projects directly depend on deep learning libraries or not. If there exists at least one import statement of deep learning libraries in deep learning programs, we then defined it as a direct dependence. Otherwise, we defined it as a transitive dependence. That is, the transitive dependence indicates that deep learning projects totally transitively depend on other deep learning projects to implement their functions.

Fig. 1. Tensorflow updating in requirement.txt.

E. Version Analysis

To record the historical dependency changes of deep learning projects towards deep learning libraries, we applied the following steps. We first cloned the studied projects from GitHub. Then, we obtained all the commits to a project's requirement.txt file (a requirement.txt file is a package management file that used to manage the versions of dependency libraries) and manually analyzed the version differences towards deep learning libraries (as exemplified in Fig. 1). Next, we extracted the version pairs towards deep learning libraries of different projects, as well as many other information, including the file path, commit sha, and timestamp. Finally, we manually determined the type of version changes (i.e., upgrade, downgrade, no change) according to the version pairs and timestamp. Notably, a project may have more than one requirement.txt files at different file paths, and sometimes a project may even have no requirement.txt file. For projects without requirement.txt file, we classify them to the type of no version changes.

Moreover, we also analyzed how often deep learning projects used the latest versions of deep learning libraries. To figure it out, we gathered the newest version statements defined in projects' requirement.txt files and manually extracted the newest version statements of deep learning libraries. It is worth noting that there may exist more than one requirement.txt files in a project, and different requirement.txt files may have different version statements. In our study, we gathered all the version statements under all requirement.txt files of those deep learning projects. Therefore, the sum of the version types may be more than the number of the deep learning projects.

For example, specifying project priya-dwivedi/Deep-Learning, its version statement in requirement.txt file with path priya-dwivedi/Deep-Learning/mask_rcnn_damage_detection/requirements.txt is $tensorflow >= 1.3.0$, while its version statements

TABLE II
CLASSIFICATION CATEGORIES OF PURPOSES.

Category of Purpose	Description	Examples
Competition	Projects that store code for the purpose of competitions, such as Bitcamp2019, Kaggle competitions, etc.	Project name: krantirk/kaggle-competition-solutions; Description: kaggle-competition-solutions; Url: https://github.com/krantirk/kaggle-competition-solutions
Knowledge Learning and Teaching	Projects that are used to learn, practice, or teach deep learning knowledge, containing code samples, demos, tutorials, etc.	Project name: microsoft/ai-edu; Description: AI education materials for Chinese students, teachers and IT professionals; Url: https://github.com/microsoft/ai-edu
Paper Experiments	Projects that are related to the research papers, such as experiment replication and algorithms reproduction of research papers.	Project name: tonybeltramelli/pix2code ; Description: pix2code: Generating Code from a Graphical User Interface Screenshot; Url: https://github.com/tonybeltramelli/pix2code
Software Development	Projects that are systems or tools of different applications, it usually includes projects of frameworks, libraries, plugins, tools, etc.	Project name: opencv/cvat; Description: Powerful and efficient Computer Vision Annotation Tool (CVAT); Url: https://github.com/opencv/cvat
Other	Projects that cannot determine their purposes or projects that cannot fit any other categories.	Project name: SelinaIrra/Diploma; Description: null; Url: https://github.com/SelinaIrra/Diploma

TABLE III
CLASSIFICATION CATEGORIES OF APPLICATIONS.

Category of Applications	Description	Examples
Code Analysis	Projects related to code processing, such as cross-language API mappings, software defect reports analyzing, etc.	Project name: bdqnghi/SAR_API_mapping; Description: FSE 2019, Learning Cross-Language API Mappings with Little Knowledge; Url: https://github.com/bdqnghi/SAR_API_mapping
Control	Projects related to the control system, such as continuous control, auto driving, robotics, etc.	Project name: aweeraman/reinforcement-learning-continuous-control; Description: Continuous Control with deep reinforcement learning where the agent must reach a moving ball with a double jointed arm; Url: https://github.com/aweeraman/reinforcement-learning-continuous-control
Efficiency Library	Projects that provide effective environment, libraries, packages, frameworks to accelerate the development of deep learning systems.	Project name: zurutech/ashpy; Description: TensorFlow 2.0 library for distributed training, evaluation, model selection, and fast prototyping; Url: https://github.com/zurutech/ashpy
Entertainment	Projects that develop games or just for fun.	Project name: michael-pacheco/deep-learning-bullet-hell-environment; Description: A reinforcement learning environment and agent for a Touhou/bullet hell inspired game: Sacred Curry Shooter; Url: https://github.com/michael-pacheco/deep-learning-bullet-hell-environment
Graph	Projects related to graph processing, such as graph representation learning, etc.	Project name: twjiang/graphSAGE-pytorch; Description: A PyTorch implementation of GraphSAGE. This package contains a PyTorch implementation of GraphSAGE; Url: https://github.com/twjiang/graphSAGE-pytorch
Image and Video	Projects related to image processing or video processing, such as image classification, image generation, image denoising, video processing, etc.	Project name: mkocabas/EpipolarPose; Description: Self-Supervised Learning of 3D Human Pose using Multi-view Geometry (CVPR2019); Url: https://github.com/mkocabas/EpipolarPose
Model Theory	Projects related to the basic model, such as model implementation, model improvement, etc.	Project name: L0SG/relational-rnn-pytorch; Description: An implementation of DeepMind's Relational Recurrent Neural Networks in PyTorch; Url: https://github.com/L0SG/relational-rnn-pytorch
Multimodality	Projects that process more than two kinds of data modalities, such as text to image, image to code, etc.	Project name: tonybeltramelli/pix2code; Description: pix2code: Generating Code from a Graphical User Interface Screenshot; Url: https://github.com/tonybeltramelli/pix2code
NLP	Projects related to text processing, such as text classification, text generation, sentiment analysis, chatbot, etc.	Project name: apcode/tensorflow_fasttext; Description: Simple embedding based text classifier inspired by fastText, implemented in tensorflow; Url: https://github.com/apcode/tensorflow_fasttext
Security	Projects related to security problems, such as SQL injection detection, DDOS detection, etc.	Project name: Aetf/tensorflow-tbcnn; Description: Tree-based Convolutional Neural Network for SQL Injection Detect; Url: https://github.com/Aetf/tensorflow-tbcnn
Time Series	Projects that used to process time-series data, such as real estate price prediction, stock prediction, etc.	Project name: Rishub21/ml_Finance; Description: Various machine learning tools to predict and analyze stock movements; Url: https://github.com/Rishub21/ml_Finance
Speech and Audio	Projects related to speech and audio processing, such as speech classification, speech recognition, music classification, etc.	Project name: IBM/MAX-Audio-Classifier; Description: Identify sounds in short audio clips; Url: https://github.com/IBM/MAX-Audio-Classifier
Other	Projects that cannot be determined the applications, projects that have no applications, or projects that cannot fit any other categories, such as a crawler, etc.	Project name: ivannz/mlss2019-bayesian-deep-learning; Description: MLSS2019 Tutorial on Bayesian Deep Learning; Url: https://github.com/ivannz/mlss2019-bayesian-deep-learning

in requirement files with paths priya-dwivedi/Deep-Learning/sentiment_classification_RNN/requirements.txt and priya-dwivedi/Deep-Learning/word2vec_skipgram/requirements.txt are all $tensorflow == 1.0.0$. We thus recorded $tensorflow == 1.0.0$ twice and $tensorflow >= 1.3.0$ once. There also exist some projects without requirement files or did not state versions in their requirement files; we defined the versions of these kinds of projects as "No-statement". In this way, we generate the distribution of versions for studied deep learning projects.

III. RESULTS

In this section, we present the results of our analysis on the sample projects to answer the four research questions.

A. Overall Impressions

Fig. 2 illustrates the box plots with the distribution of the number of stars, the number of contributors, and project size of the studied projects. Results show that PyTorch-dependent projects are more popular than Theano-dependent and Tensorflow-dependent projects (4 vs. 3 vs. 2, median measures), but have fewer contributors than Theano-dependent and Tensorflow-dependent projects (1 vs. 2 vs. 1, median measures), and also have smaller project size than Theano-dependent and Tensorflow-dependent projects (4,052 vs. 8,112 vs. 8,248, median measures), which is aligned with previous studies that PyTorch-dependent projects are more lightweight and easier to implement and use [4].

To investigate whether the distributions of different groups are statistically significant, we performed the Wilcoxon Signed-Rank test [39] at the confidence level of 95% and computed Cliff's delta [11] to show the effect size of the difference. Indeed, the distributions of different groups on stars and contributors are statistically different, and the effect sizes are non-negligible, while the distributions of different groups on project size have no statistical difference.

B. RQ1: What are the purposes and applications of open source projects that depend on Tensorflow, PyTorch, and Theano?

In this section, we present and compare the distributions of 5 purpose categories and 13 application categories on Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects, respectively.

1) Purpose Category Distribution: Fig. 3 demonstrates the distributions of purpose categories on Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects. It shows that the software development purpose is accompanied with the highest number of occurrences for Tensorflow-dependent projects, and the knowledge learning and teaching purpose has the second-highest number of occurrences. For PyTorch-dependent projects, paper experiments purpose accounts for the overwhelming majority. As for Theano-dependent projects, software development and paper experiments purposes take up more than half of the total.

We can find that the commonality is that **the vast majority of the purposes of projects distribute at software development, paper experiments, and knowledge learning and teaching, which accounts for at least 86% of the total**. The discrepancy is that Tensorflow-dependent projects concentrate more on software development, while PyTorch-dependent projects concentrate more on paper experiments, which is in line with the previous study that Tensorflow does well in industrial production capabilities, while PyTorch deeply plows the research community [4]. To examine whether the distribution of purposes of open source projects that depend on different deep learning libraries are statistically different, we applied Kolmogorov-Smirnov test [28], [38]. The null hypothesis is that open source projects that depend on different deep learning libraries have the same distribution of each purpose category. Results show that there exists no significant difference in the distribution.

2) Application Category Distribution: Fig. 4 illustrates the distributions of application categories on Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects. Fig. 4(a) shows that the most common application of Tensorflow-dependent projects is image and video, which takes up 28% of the total, and the NLP application ranked at the second with 14% of the total. One potential explanation for this finding may be due to that the image and video and NLP applications comprise various sub-applications and each sub-application has attracted many users involved, e.g., the image and video application encompasses face recognition (e.g., accessai/access-face-vision), semantic

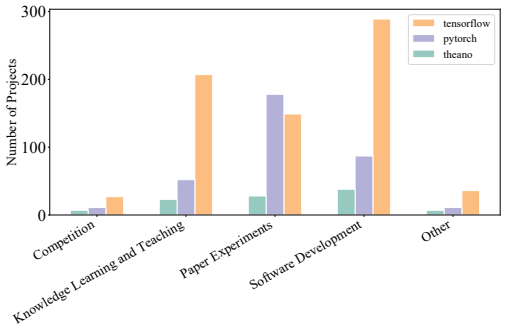


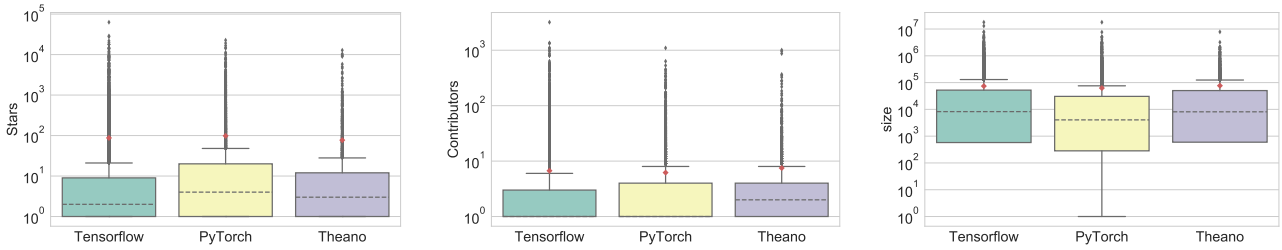
Fig. 3. Distributions of purpose categories of Tensorflow/PyTorch/Theano-dependent projects.

segmentation (e.g., Acciorocketships/FCN), human pose estimation (e.g., MrEliptik/HandPose), and healthcare (e.g., saigerutherford/fetal-code), while the NLP application contains chatbot (e.g., Chriszhangmw/ChatBots), sentiment analysis (e.g., lixin4ever/BERT-E2E-ABSA), and name entity recognition (e.g., uhh-lt/microNER).

Simultaneously, Fig. 4(b) for PyTorch-dependent projects and Fig. 4(c) for Theano-dependent projects also reveal that the most common two applications are image and video and NLP. For Tensorflow-dependent projects, we can also observe that the least common applications are code analysis and graph, which only takes up 2% of the total, while the least common applications for PyTorch-dependent projects are code analysis, graph, and multimodality, which only accounts for 3% of the total. As for the Theano-dependent projects, code analysis, graph, and entertainment applications are the least common applications, which account for 3% of the total. A reasonable explanation for these findings may be attributed to that the code analysis and graph applications have limited sub-applications and fewer users are engaged in the two domains.

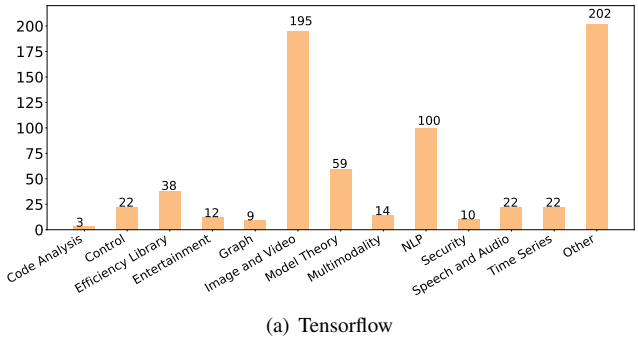
As a result, our analysis reveals that **the image and video, NLP, model theory, and efficiency library applications are the four most common applications for Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects**, which accounts for 55%, 71%, and 51% of the total, respectively. Meanwhile, **the code analysis and graph applications are the two least common applications for those projects**. It is also worth noting that Tensorflow-dependent projects have the widest distribution at all the applications, while PyTorch-dependent projects have no distribution at security and time series applications, and Theano-dependent projects have no distribution at multimodality and security applications. It may imply that Tensorflow does better at multimodality, security, and time series applications.

We then performed Kolmogorov-Smirnov test [28], [38] to check whether the distribution of applications of open source projects that depend on different deep learning libraries is statistically different. The null hypothesis is that open source projects that depend on different deep learning libraries have the same distribution of application categories. As a result, we observe that there exists a significant difference in the distribution between Tensorflow-dependent projects and Theano-dependent projects. Apart from this, there exists no

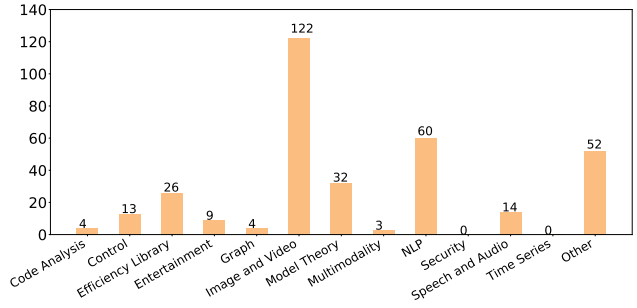


(a) Stars (b) Contributors (c) Size

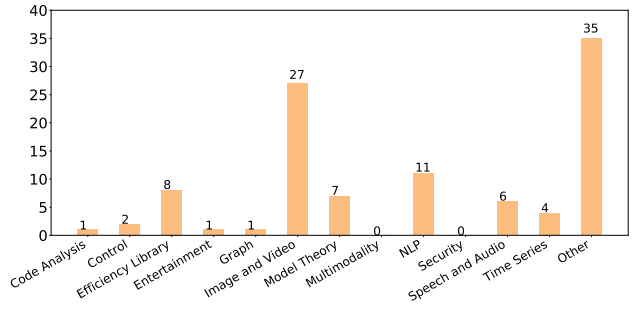
Fig. 2. Distributions of (a) stars, (b) contributors, and (c) size of studied projects that depend on different deep learning libraries.



(a) Tensorflow



(b) PyTorch



(c) Theano

Fig. 4. Distributions of application categories of Tensorflow/PyTorch/Theano-dependent projects.

other significant difference in the distribution.

C. RQ2: To what extent do deep learning projects depend on Tensorflow, PyTorch, and Theano?

Deep learning projects can directly or transitively depend on deep learning libraries to implement their algorithms. To understand the extent to which they depend on deep learning libraries, we discuss the dependency degrees of deep learning

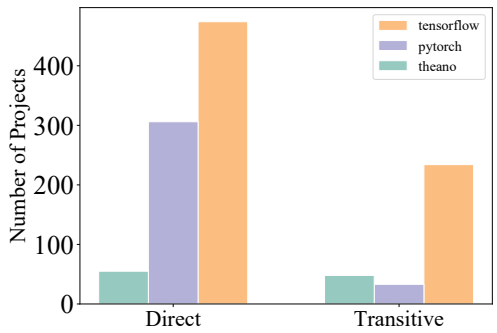
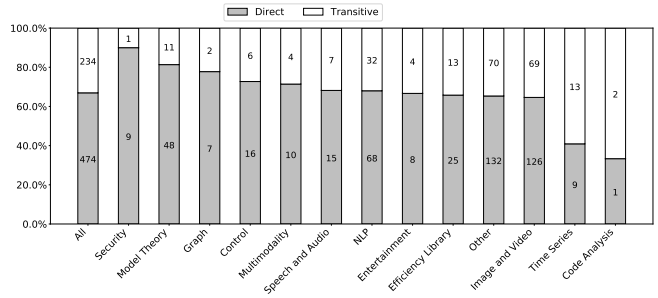


Fig. 5. Distribution of direct and transitive dependencies of Tensorflow/PyTorch/Theano-dependent projects.

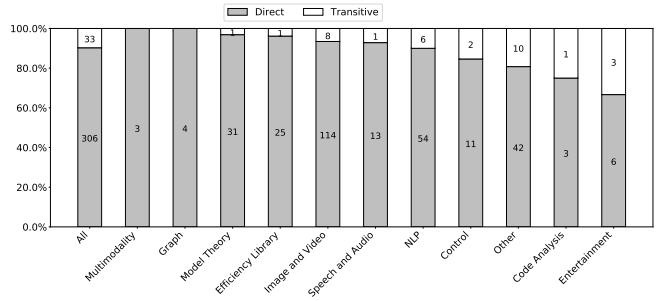
projects on Tensorflow, PyTorch, and Theano, respectively. In this section, we introduce the distribution of the number of direct and transitive dependencies, and analyze how other factors influence the distributions.

Fig. 5 shows the distribution of the number of direct and transitive dependencies on Tensorflow, PyTorch, and Theano. We can find that **all deep learning projects have a higher proportion of direct dependencies than transitive dependencies**, Tensorflow-dependent projects show 103% more direct dependencies comparing to transitive dependencies, PyTorch-dependent projects show 827% more direct dependencies comparing to transitive dependencies, while Theano-dependent projects only show 15% more direct dependencies comparing to transitive dependencies.

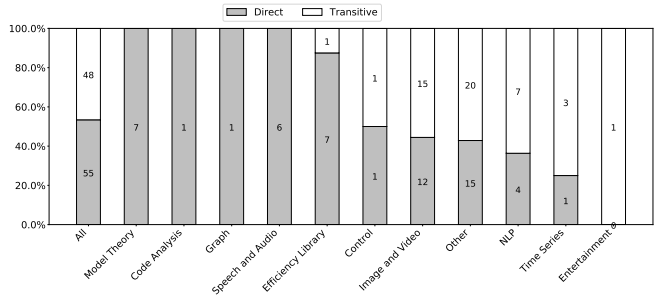
Then, we present the percentage of deep learning projects with various application domains across different dependency degrees, as exemplified in Fig. 6. We notice that Tensorflow-dependent projects with security, model theory, and graph applications are the top three groups that directly rely on Tensorflow, while projects with code analysis, time series, and image and video applications are the top three groups that transitively rely on Tensorflow. Meanwhile, PyTorch-dependent projects with multimodality, graph, and model theory applications are the top three groups that directly rely on PyTorch, while projects with entertainment, code analysis, and control applications are the top three groups that transitively rely on PyTorch. As for Theano, projects with model theory, code analysis, and graph applications are the top three groups that directly rely on Theano, while projects with entertainment, time series, and NLP applications are the top three groups that transitively rely on Theano.



(a) Tensorflow



(b) PyTorch



(c) Theano

Fig. 6. Distributions of Tensorflow/PyTorch/Theano-dependent projects of dependency degrees across different application categories.

We also performed the Kolmogorov-Smirnov test [28], [38] to examine whether the distribution of dependence degree on various applications is statistically significant. The null hypothesis is that deep learning projects with different applications have the same distribution of direct or transitive dependence on deep learning libraries. Consequently, we observe that there exists a significant difference in the distribution between Tensorflow-dependent projects and Theano-dependent projects as for direct dependence and transitive dependence, and there also exists a significant difference in the distribution between Tensorflow-dependent projects and Pytorch-dependent projects as for the transitive dependence. Apart from this, there is no other significant differences.

D. RQ3: How are the update behaviors of deep learning projects that depend on Tensorflow, PyTorch, and Theano?

Developers may update deep learning libraries for various reasons. In this RQ, we aim to understand how different deep learning projects manage their dependencies on deep learning libraries, and why and how difficult they update the dependencies.

TABLE IV
THE STATISTICS OF THE UPDATE BEHAVIORS FOR DEEP LEARNING PROJECTS THAT DEPEND ON TENSORFLOW, PYTORCH, AND THEANO.

	Upgrade	Downgrade	No Change	Update Time Lag	Release Time Lag
Tensorflow	79	9	645	143	104
PyTorch	41	5	313	117	81
Theano	1	1	101	0	379

* *The update time lag means the lag between dependency updates, while the release time lag means the lag between library release time and dependency update time.*

1) Type of Version Changes: **There exist three types of version changes – upgrade, downgrade, and no change**. Tensorflow, PyTorch, and Theano have been upgraded for 79 (11%), 41 (11%), and 1 (1%) times, respectively. Meanwhile, there also exist some downgrade behaviors, Tensorflow, PyTorch, and Theano have been downgraded for 9 (1%), 5 (1%), and 1 (1%) times, respectively. We can find that there only a small percentage of projects have upgraded deep learning libraries.

Note that the sum of the three types of version changes is larger than the number of projects, which is due to that some projects may upgrade deep learning libraries for more than one time. For example, the Tensorflow-dependent project spotify/spotify-tensorflow has upgraded Tensorflow versions for 9 times from Tensorflow 1.2.0 to Tensorflow 1.15.x during 15, September 2017 to 30, October 2019.

To investigate why these projects upgraded or downgraded deep learning libraries, we manually analyzed the commits of these projects to find the update reasons. Results show that Tensorflow-dependent projects upgrade Tensorflow out of the following reasons: 1) addressing critical severity vulnerability, 2) merging an automatic pull request derived by an automatic dependency update tool – Dependabot to update the outdated or insecure libraries, 3) upgrading to ensure their Tensorflow dependency version less than 2.0, for that Tensorflow 2.0 has some pretty big API changes and they do not want to use it, 4) upgrading the Tensorflow version to include more features, e.g., the project alessiamarcolini/MyQ upgraded the Tensorflow version from 1.12.0 to 1.13.1 to incorporate with GPU support, etc, 5) upgrading the Tensorflow version to make it compatible with existing frameworks or libraries, e.g., the project IBM/MAX-Audio-Classifier upgraded the Tensorflow version from 1.12.2 to 1.13.1 to ensure Tensorflow work on ARM. PyTorch-dependent projects upgrade PyTorch version to make it compatible with existed frameworks or libraries, e.g., the project lidq92/CNNIQA upgraded the PyTorch version from 0.4.0 to 0.4.1 to satisfy the minimum need of ignite library.

Besides, Tensorflow-dependent projects downgrade Tensorflow to stop using Tensorflow 2.0, PyTorch-dependent projects downgrade PyTorch to address the issues that reported the current version could not work, while Theano-dependent projects downgrade Theano due to that they believe that more recent versions of Theano may cause conflicts.

2) Time Lag: We find that, on average, **the update time lag of Tensorflow-dependent projects is longer than PyTorch-**

dependent and Theano-dependent projects. They had about 22% longer lag between dependency updates comparing to PyTorch-dependent projects. The longest update time lag of Tensorflow-dependent projects is 853 days, which belongs to the project chxj1992/captcha_cracker, this project upgraded the Tensorflow version from Tensorflow 1.1.0 to Tensorflow 1.12.2. The longest update time lag of PyTorch-dependent projects is 480 days, which belongs to the project ixaxaar/pytorch-dnc, this project upgraded the PyTorch version from PyTorch 0.2.0 to PyTorch 1.0.1. They all have a big update as for the deep learning library version. To examine whether the distribution of update time lag is significantly different, we performed the Wilcoxon Signed-Rank test at a p-value of 0.05 [39] and computed Cliff's delta [11]. Results show that there exists no significant difference.

Besides, we also find that on average, **the release time lag of Tensorflow-dependent projects is longer than PyTorch-dependent projects**, they had about 28% longer time lag comparing to PyTorch-dependent projects. Interestingly, **the release time lag of Theano-dependent projects is longer than Tensorflow-dependent and PyTorch-dependent projects**, which had about 264% and 368% longer lag than them. To examine whether the distribution of release time lag is significantly different, we performed the Wilcoxon Signed-Rank test at a p-value of 0.05 [39] and computed Cliff's delta [11] again. Results show that the release time lag of Tensorflow-dependent projects is statistically significantly longer than PyTorch-dependent projects, and the effect size is medium. Besides, the release time lag of Theano-dependent projects is statistically significantly longer than Tensorflow-dependent projects, and the effect size is large.

E. RQ4: How often do deep learning projects use the latest versions of Tensorflow, PyTorch, and Theano?

Different deep learning projects are apt to use different versions of deep learning libraries. To comprehend how often deep learning projects use the latest versions of deep learning libraries, we present the distributions of dependency versions of Tensorflow, PyTorch, and Theano.

1) Version Distribution: Our results are displayed in Fig. 7. We can notice that **Theano-dependent projects had the highest percentage of the latest versions**. That is, 20% of the Theano-dependent projects used Theano 1.0, vs. 3% for Tensorflow 2.0, and 0.3% for PyTorch 1.4. A reasonable explanation may be attributed to that the official release dates of the latest versions of Tensorflow 2.0 and PyTorch 1.4 are too close that the latest versions have not been widely adopted. As Tensorflow 2.0 was released in September 2019 [5], the latest version of PyTorch 1.4 was released in January 2020 [2], while the latest version Theano 1.0 was released in November 2017.

Besides, we also observe that most Tensorflow-dependent projects use the versions of Tensorflow 1.0, Tensorflow 1.12, and Tensorflow 1.13, and most PyTorch-dependent projects use the versions of PyTorch 0.4 and PyTorch 1.0. As for Theano-dependent projects, most of them use the latest versions of Theano 0.8, Theano 0.9, and Theano 1.0. An intriguing phenomenon is that **Tensorflow-dependent and PyTorch-dependent projects are apt to use a recent but not the latest versions**. It may be due to that Tensorflow and PyTorch have way more versions and are releasing new versions frequently, more versions lead to more choices and the newest versions are apt to exist big API changes or compatibility issues, thus users are prone to select a recent but not the latest version to avoid various unexpected problems.

Moreover, another reason may be attributed to that many users have used TensorFlow and PyTorch for a long time. Despite TensorFlow and PyTorch released the latest version, they are unwilling to spend extra time and energy to upgrade, for that the upgrading process always involves various perspectives and many extra efforts (e.g., the syntactic and stylistic changes). Similar views were expressed by Kula et al. [25], i.e., they claimed that "developers do not prioritize updates for that they cited it as an added effort to be performed in their spare time."

Simultaneously, It is also worth noting that at least one-fifth of the projects have no statement (including no requirement files or no version statements in requirement files). 34%, 35%, and 21% of the Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects have no statement, respectively. Results indicate that there are too many deep learning projects that have no version management of deep learning libraries, which is detrimental to themselves and those projects that depend on them, and are prone to yield crashes or security issues.

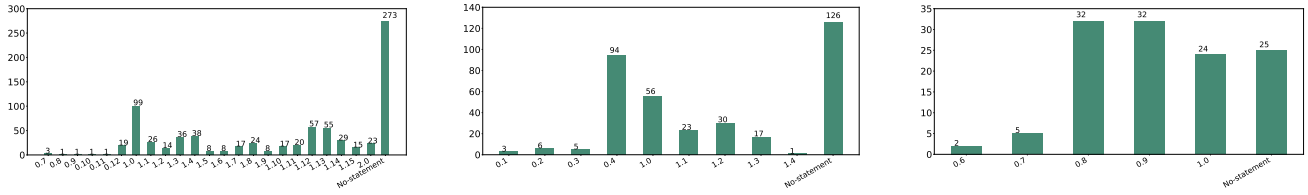
IV. Discussion

In this section, we discuss and provide some practical implications for deep learning library developers, deep learning library users, and software engineering researchers, and also discuss the threats to validity.

A. Implications

Implications for developers of deep learning libraries. Our results reveal that some deep learning users upgrade Tensorflow to ensure their dependency version less than 2.0, they avoid using Tensorflow 2.0 for that Tensorflow 2.0 has some pretty significant API changes. Although Tensorflow communities have provided an automatic upgrade script (tf_upgrade_v2) to help migrate TensorFlow from 1.x to TensorFlow 2.0, there are still very few users adopt it. A reasonable explanation may be due to that the script cannot perform the syntactic and stylistic changes [1], so that it is still time-consuming for users to perform a big update. Deep learning library developers should develop a more complete automatic tool or script to improve the efficiency of version migration.

Besides, they should not only explain why an update should be made to a deep learning project, but also take into account confidence measures that users can use to estimate the risk of performing a big update. For instance, they can calculate how many upgrades to TensorFlow 2.0 are performed and how many downgrades from TensorFlow 2.0 are tracked to give a confidence score for upgrade success.



(a) Tensorflow (b) PyTorch (c) Theano

Fig. 7. Distributions of dependency versions of Tensorflow/PyTorch/Theano-dependent projects.

Implications for users of deep learning libraries. Our findings also reveal that some users upgrade deep learning libraries to address critical severity vulnerabilities or to make it compatible with existed frameworks or libraries. However, there exists a relatively large ratio of transitive dependencies on deep learning libraries of deep learning projects. For this kind of projects, users cannot directly manage the dependencies of deep learning libraries, which may result in security vulnerabilities or code crashes due to the issues existed in deep learning libraries of dependent projects. Therefore, deep learning users may need to utilize tools such as Software Composition Analysis (SCA) [9] products (e.g., Foo et al.'s SCA design [16], Veracode [3], etc.) to deal with transitive dependencies of vulnerable versions of deep learning libraries. SCA can automatically identify the vulnerable versions of the dependencies used in a project, so that users can transitively depend on deep learning libraries and keep away with security issues.

Implications for software engineering researchers. We find that there exist various purposes of deep learning projects, therefore, researchers should also consider the purposes to fit their research needs when they select deep learning projects to analyze. Meanwhile, the code analysis and graph applications are the least two common applications for Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects, which indicates that there still too few works that have been open sourced in these two domains. Researchers in these two domains should pay more efforts to propagate and promote their works to attract more attention, e.g., open source their works in GitHub and introduce their works thoroughly in the readme files. By opening source their works in GitHub, they can attract more contributors and researchers and provide avenues for them to contribute, which can bring a more extensive development of these two domains.

Besides, As most of the Tensorflow-dependent and PyTorch-dependent projects are apt to use a recent but not the latest versions, and sometimes, even if the projects have been upgraded to the latest version, they will still downgrade it back. Developers and researchers should make more efforts to provide more details about the update process, such as providing empirical evidence of success, e.g., other users with similar versions have updated to the latest versions and only spent fifteen minutes. Meanwhile, developers and researchers can also provide thorough tutorials to help users effectively understand the update changes of their libraries.

Moreover, our results also reveal that there exist many projects that do not have version statements, some of them even do not have requirement file. Users and researchers should pay more attention to the version management and form a good habit, including creating requirement files and managing the library versions. Besides, developers and researchers can also provide automatic tools to help users generate requirement files and record versions automatically.

B. Threats to Validity

Internal Threats. We gathered open source projects that depend on Tensorflow, PyTorch, and Theano via the dependency graph provided by GitHub API. Since we are the first to use the dependency graph to extract the data, there is no guarantee that this API can generate correct results. To mitigate this threat, we verified the correctness of the extracted data by manually checking the sample projects, and results claimed the correctness of our dataset.

Another threat can be attributed to the trustworthiness of our sample projects. To ameliorate this threat, we first removed projects that were forked, non-starred, and deleted. After that, we performed a sampling process on the filtered projects and stratified selection process according to the popularity. In this way, we guaranteed the quality of the dataset and the consistency of the data structure to the maximum extent.

Ultimately, the third internal threat related to the classification categories. To mitigate this threat, we adapted the purpose categories based on Kalliamvakou et al.'s study [24] and adapted the application categories on top of Liu et al. and Deng et al.'s studies. In this process, two experts from natural language processing and computer vision domains also involved our study and determined categories jointly with all the authors following a card sorting approach.

External Threats. The results in our paper may not generalize to all deep learning libraries. Although we studied a large sample of open source projects that depend on Tensorflow, PyTorch, and Theano, results of this paper may not extend to all deep learning libraries. However, due to that the three deep learning libraries are all typical and popular, we believe that our findings can also bring some revelations to developers, users, and researchers that use other deep learning libraries. Besides, another external threat involves that the findings in this paper may not adapt to proprietary DL systems, since we only obtained the open source projects in GitHub, but not gathered the proprietary projects, where proprietary projects may have different dependency management patterns with open source projects.

V. Related Work

In this section, we review the related works in two aspects: studies on deep learning (DL) systems and studies on dependency networks. To the best of our knowledge, we initiate the first step towards the dependency management of deep learning systems.

A. Studies of Deep Learning Systems

The deep learning ecosystem has grown in leaps and bounds in the past few years, which has led to a tremendous amount of research effort. Du et al. [14] made a quantitative analysis of RNN-based DL systems. They proposed a general-purpose quantitative analysis framework DeepStellar to characterize the internal behaviors of RNN-based DL systems. Guo et al. [17] conducted a systematic study on four deep learning frameworks (Tensorflow, PyTorch, CNTK, and MXNET) and two platforms (mobile and web) to characterize the impacts of current DL frameworks and platforms on DL software development and deployment processes. Han et al. [19] applied Latent Dirichlet Allocation (LDA) to derive and compare the discussion topics concerning three popular deep learning frameworks (Tensorflow, PyTorch, and Theano) on two platforms (GitHub and Stack Overflow).

Moreover, the study of the performance of deep learning frameworks also has been subject to numerous investigations. Ha et al. [18] proposed an approach called DeepPerf to predict performance values of highly configurable software systems. They performed it by using a deep feedforward neural network (FNN) combined with a sparsity regularization technique. Liu et al. [26] presented design considerations, metrics, and challenges towards developing an effective benchmark for DL software frameworks, and conducted a comparative study on three popular DL frameworks, namely, TensorFlow, Caffe, and Torch. Shams et al. [35] attempted to analyze the performance of different deep learning frameworks (Caffe, TensorFlow, and Apache SINGA) in different hardware environments. To investigate it, they compared the time per training iteration and the number of images trained within a millisecond. Bahrampour et al. [7] performed a comparative study of four deep learning frameworks, namely Caffe, Neon, Theano, and Torch on three aspects, which are extensibility, hardware utilization, and speed.

In addition to the above studies on deep learning frameworks, there also exist many studies focus on bug detection and localization of deep learning frameworks. For instance, Zhang et al. [41] studied deep learning applications built on top of TensorFlow and collected their program bugs to determine the root causes and symptoms of these bugs. Islam et al. [23] studied 2,716 high-quality posts from Stack Overflow and 500 bug fixing commits from Github to understand the bugs types, root causes, impacts, bug-prone stages as well as the common antipatterns in buggy software. Besides, Pham et al. [33] proposed a new approach - CRADLE to find and localize bugs in DL software libraries and performed it by cross-checking multiple backends.

B. Studies of Dependency Networks

There also existed many research works towards dependency and maintainability issues across different programming language ecosystems. For example, Wittern et al. [40] studied the evolution of the npm JavaScript library ecosystem and analyzed their characteristics such as dependencies, popularity, version distribution, etc. While Decan et al. [12] captured the growth, changeability, reusability, and fragility of dependency networks on seven packaging ecosystems: Cargo for Rust, CPAN for Perl, CRAN for R, npm for JavaScript, NuGet for the .NET platform, Packagist for PHP, and RubyGems for Ruby. Kula et al. [25] conducted an empirical study on library migration to investigate the extent to which developers update their library dependencies.

Although there have been subject to many efforts on dependency networks across various programming language ecosystems, there have not yet existed studies to study dependency networks of deep learning systems. Our study can shed light on the dependency management for deep learning ecosystems and provide new research directions such as how deep learning systems migrate to new versions of libraries.

VI. Conclusion

In this paper, we take the first step to perform a comparative study to explore the dependency networks of deep learning libraries, i.e., Tensorflow, PyTorch, and Theano. In our study, we investigated 708 Tensorflow-dependent projects, 339 PyTorch-dependent projects, and 103 Theano-dependent projects to identify the purposes and applications (RQ1), dependency extents (RQ2), and update behaviors (RQ3) as well as dependency versions (RQ4) of deep learning projects. Our analysis reveals that there exist some commonalities as for the purpose distributions, application distributions, and dependency extents of Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects. There also have some discrepancies as for the update behaviors, update reasons, update time lag, and dependency version distributions of Tensorflow-dependent, PyTorch-dependent, and Theano-dependent projects.

In the future, we plan to encompass more deep learning libraries and incorporate proprietary projects to expand the generalization of our results. Moreover, we also encourage further studies to analyze more additional questions and extend our work, e.g., to analyze the relationship between direct/transitive dependencies and upgrade/downgrade behaviors.

Acknowledgment

This research was partially supported by the National Key Research and Development Program of ChinaNo. 2017YFB1400601), National Science Foundation of China (No. 61772461), Natural Science Foundation of Zhejiang Province (No. LR18F020003) and also was supported by Alibaba-Zhejiang University Joint Research Institute of Frontier Technologies.

REFERENCES

[1] "Automatically upgrade code to tensorflow 2," https://www.tensorflow.org/guide/upgrade.

[2] "Pytorch 1.4 released, domain libraries updated," https://pytorch.org/blog/pytorch-1-dot-4-released-and-domain-libraries-updated/, accessed: 2020-01-15.

[3] "Software composition analysis veracode." https://www.veracode.com/.

[4] "The state of machine learning frameworks in 2019," shorturl.at/lsu26, accessed: 2019-10-10.

[5] "Tensorflow 2.0 is now available!" https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html, accessed: 2019-09-30.

[6] D. Acuna, A. Kar, and S. Fidler, "Devil is in the edges: Learning semantic boundaries from noisy annotations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 075–11 083.

[7] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of caffe, neon, theano, and torch for deep learning," 2016.

[8] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[9] Y. Chen, A. E. Santosa, A. Sharma, and D. Lo, "Automated identification of libraries from vulnerability data," 2020.

[10] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[11] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[12] A. Decan, T. Mens, and P. Grosjean, "An empirical comparison of dependency network evolution in seven software packaging ecosystems," *Empirical Software Engineering*, vol. 24, no. 1, pp. 381–416, 2019.

[13] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[14] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Deepstellar: model-based quantitative analysis of stateful deep learning systems," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 477–487.

[15] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[16] D. Foo, J. Yeo, H. Xiao, and A. Sharma, "The dynamics of software composition analysis," *arXiv preprint arXiv:1909.00973*, 2019.

[17] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 810–822.

[18] H. Ha and H. Zhang, "Deepperf: performance prediction for configurable software with deep sparse neural network," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1095–1106.

[19] J. Han, E. Shihab, Z. Wan, S. Deng, and X. Xia, "What do programmers discuss about deep learning frameworks."

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[21] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[22] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

[23] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 510–520.

[24] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 92–101.

[25] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *Empirical Software Engineering*, vol. 23, no. 1, pp. 384–417, 2018.

[26] L. Liu, Y. Wu, W. Wei, W. Cao, S. Sahin, and Q. Zhang, "Benchmarking deep learning frameworks: Design considerations, metrics and beyond," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1258–1269.

[27] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[28] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[29] S. Mirhosseini and C. Parnin, "Can automated pull requests encourage software developers to upgrade out-of-date dependencies?" in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 84–94.

[30] M. Nica, "Optimal strategy in guess who?: Beyond binary search," *Probability in the Engineering and Informational Sciences*, vol. 30, no. 4, pp. 576–592, 2016.

[31] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz *et al.*, "Attention u-net: Learning where to look for the pancreas," *Conference on Medical Imaging with Deep Learning (MIDL)*, 2018.

[32] J. Petersen, P. F. Jäger, F. Isensee, S. A. Kohl, U. Neuberger, W. Wick, J. Debus, S. Heiland, M. Bendszus, P. Kickingereder *et al.*, "Deep probabilistic modeling of glioma growth," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2019, pp. 806–814.

[33] H. V. Pham, T. Lutellier, W. Qi, and L. Tan, "Cradle: cross-backend validation to detect and localize bugs in deep learning libraries," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1027–1038.

[34] R. Pryzant, R. D. Martinez, N. Dass, S. Kurohashi, D. Jurafsky, and D. Yang, "Automatically neutralizing subjective bias in text," *arXiv preprint arXiv:1911.09709*, 2019.

[35] S. Shams, R. Platania, K. Lee, and S.-J. Park, "Evaluation of deep learning frameworks over different hpc architectures," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1389–1396.

[36] D. Spencer, *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[37] Z. Wan, D. Lo, X. Xia, and L. Cai, "Bug characteristics in blockchain systems: a large-scale empirical study," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 413–424.

[38] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, and X. Yang, "Perceptions, expectations, and challenges in defect prediction," *IEEE Transactions on Software Engineering*, 2018.

[39] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

[40] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the javascript package ecosystem," in *Proceedings of the 13th International Conference on Mining Software Repositories*, 2016, pp. 351–361.

[41] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang, "An empirical study on tensorflow program bugs," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 129–140.