gDefect4DL: A Dataset of General Real-World Deep Learning Program Defects

Yunkai Liang
liang1996@tju.edu.cn
Tianjin University
China

Yun Lin*
dcsliny@nus.edu.sg
National University of Singapore
Singapore

Xuezhi Song
songxuezhi@fudan.edu.cn
Fudan University
China

Jun Sun
junsun@smu.edu.sg
Singapore Management University
Singapore

Zhiyong Feng
zyfeng@tju.edu.cn
Tianjin University
China

Jin Song Dong
dcsdjs@nus.edu.sg
National University of Singapore
Singapore

ABSTRACT

The development of deep learning programs, as a new programming paradigm, is observed to suffer from various defects. Emerging research works have been proposed to detect, debug, and repair deep learning bugs, which drive the need to construct the bug benchmarks. In this work, we present *gDefects4DL*, a dataset for *general* bugs of deep learning programs. Comparing to existing datasets, *gDefects4DL* collects bugs where the root causes and fix solutions can be well generalized to other projects. Our general bugs include deep learning program bugs such as (1) violation of deep learning API usage pattern (e.g., the standard to implement cross entropy function $y \cdot log(y)$, $y \rightarrow 0$, without NaN error), (2) shape-mismatch of tensor calculation, (3) numeric bugs, (4) type-mismatch (e.g., confusing similar types among numpy, pytorch, and tensorflow), (5) violation of model architecture design convention, and (6) performance bugs.

For each bug in *gDefects4DL*, we describe why it is general and group the bugs with similar root causes and fix solutions for reference. Moreover, *gDefects4DL* also maintains (1) its buggy/fixed versions and the isolated fix change, (2) an isolated environment to replicate the defect, and (3) the whole code evolution history from the buggy version to the fixed version. We design *gDefects4DL* with extensible interfaces to evaluate software engineering methodologies and tools. We have integrated tools such as ShapeFlow, DEBAR, and GRIST. *gDefects4DL* contains 64 bugs falling into 6 categories (i.e., API Misuse, Shape Mismatch, Number Error, Type Mismatch, Violation of Architecture Convention, and Performance Bug). *gDefects4DL* is available at https://github.com/llmhyy/defects4dl, its online web demonstration is at http://47.93.14.147:9000/bugList, and the demo video is at https://youtu.be/0XtaEt4Fhm4.

*Corresponding author.

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9223-5/22/05…$15.00
https://doi.org/10.1145/3510454.3516826

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

datasets, neural networks, deep learning, defects, bugs

ACM Reference Format:
Yunkai Liang, Yun Lin, Xuezhi Song, Jun Sun, Zhiyong Feng, and Jin Song Dong. 2022. gDefect4DL: A Dataset of General Real-World Deep Learning Program Defects. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3510454.3516826

1 INTRODUCTION

Deep learning models and programs have been widely applied in various software systems, which increases the demand of the techniques for their testing, debugging, and repair [12, 14, 15, 19]. Many researchers have constructed various bug datasets for deep learning programs, for the purpose of empirical studies [3, 17, 18] and the development of techniques to automate bug-related tasks [12, 14, 15, 19]. The collected bugs are classified into different fix patterns (e.g., change loss function, optimizer, neural architecture, etc) [4], job failure types (e.g., corrupt data, syntax error, etc.) [3, 17], symptoms (e.g., low efficiency), and root causes (e.g., incorrect model hyper-parameter as learning rate) [18].

Despite the effort, existing deep learning bug collections lack clear *specification* to understand why the bugs happen. For example, some fixes in the dataset [18] are to load an additional pretrained model or add one more layer of activation function. Without well-defined specification, it is hard to know whether such a fix (as well as the root cause of the bug) can be generalized to the deep learning programs in other projects. We observe that the similar problem also applies to datasets proposed in [3] and [4]. Listing 1 and Listing 2 show a project-specific and a general deep learning program bug respectively.

In this work, we introduce *gDefects4DL*, a dataset for *general* deep learning bugs, which aims to facilitate research on general bug fixing techniques for deep learning programs. We ensure the bugs are general in two folds. First, we specify each bug with the specification it violates, such as implicit API precondition and type/dimension compatibility. Second, we provide each bug with a *support level*, indicating how many similar bugs share similar root causes and

```
for batch in data.batch_q_iter:
    feed_dict = {inputs: batch}
    results = model.infer(sess, feed_dict)
    bs_infer = results['output']['bs_infer']
    batch_hyps = gen_text(bs_infer, ...)
    + if beam_width > 0:
    bs_turn = dict(question=question, answer=batch_hyps[1][i])
```

Listing 1: A bug fix reported in [17], which is caused by project-specific logical error. The bug and fix are difficult to generalize to other projects.

```
with tf.name_scope("cost"):
    if loss_func == 'cross_entropy':
        cost = - tf.reduce\_mean(ref\_input *
            - tf.log(model_output)
            + tf.log(tf.clip_by_value(model_output, 1e-10, float('inf')))
        ...
        self.cost = cost
```

Listing 2: A bug fix in *gDefects4DL* for a numeric error, which makes the *log*() function undefined with the input variable *model_output* is approaching 0. The fix solution by applying a *tf.clip_by_value()* function is popular in multiple projects.

fix solutions in the dataset. Overall, we construct *gDefects4DL* with the following goals:

- **Generality:** The bugs in *gDefects4DL* are selected based on how general the deep-learning program defects are. When constructing the dataset, we avoid the application-specific defects whose specification can hardly be generalized.
- **Authenticity:** Each defect in *gDefects4DL* is selected from a real-world closed issue on a popular Github repository. Thus, each bug is naturally documented with all the corresponding discussion and code evolution history.
- **Isolated Environment:** *gDefects4DL* also isolates the runtime environment of each defect as both Conda and Docker implementation. Moreover, we enhance a test case for each bug to distinguish the buggy and the fixed version.
- **Tool Extension:** *gDefects4DL* supports a variety of programmable interfaces, potentially integrable to existing and future deep-learning based software engineering tools. *gDefects4DL* now integrates tools as ShapeFlow [12], DEBAR [19] and GRIST [15].

gDefects4DL consists of 64 deep-learning program defects falling into 6 categories. The source code, demo video, and demo website is available at https://github.com/llmhyy/defects4dl, https://youtu.be/0XtaEt4Fhm4, and http://47.93.14.147:9000/bugList.

2 DATABASE CONSTRUCTION

Figure 1 shows our workflow to collect the deep-learning program defects from the popular deep-learning Github code repositories. We consider a Github repository as a deep learning project if it uses a popular framework like Tensorflow, PyTorch, or Keras. We consider a Github repository as popular if the number of either its stars, forks, or watches is larger than a threshold (e.g., 100). The whole process starts with a closed issue on a popular deep-learning repository, and isolates a replicable deep-learning program defect consisting of its meta information (including defect type, error type,

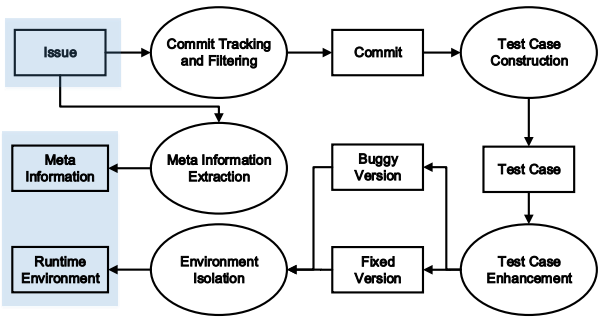

Figure 1: Work flow of *gDefects4DL* database construction

description, etc.) and runtime environment (in the forms of Conda and Docker container) to replicate the bug and its fix.

Commit Tracking and Filtering. Given an issue, we first track the commit to close the issue. In this work, we build such traceability through two sources. First, we analyze the issue discussion threads for the potential Git commit id. Second, we search through the whole git commit history and report the commit with message containing the issue id. By this means, for each issue, we can have a set of candidate commits through the automated approaches. To minimize the manual effort, we only manually check the issue reported with only one candidate commit by comparing the code with the issue description.

Once the issue-commit relation is confirmed, then we search through the issues to see whether the patch (i.e., commit) to fix it is generalizable in two ways. On one hand, we define the similarity between the ASTs of two fixes (i.e., patches) and their context. If the similarity is higher than a threshold, we further manually check whether the defect fixed by the commit is general and group the bugs with similar root causes and fix solutions. The group size is the *support level* of the general bug. On the other hand, we manually sample bugs with no similar bugs (i.e., support level is 1) to observe their generality potential. Specifically, two of the authors first select bugs with high generality potential and ask an expert with 3 years of experience of deep learning program development to confirm their generality. As a result, we collect 64 general bugs, falling into the categories in Table 1.

In contrast to those bugs in dataset like [1], [3], [4] and [18], our collected bugs and their fix solution in *gDefects4DL* can largely be generalized to other deep learning projects.

Test Case Construction and Enhancement. Once a commit c to fix an issue is confirmed, we will manually construct a test case t so that t can pass in the version of c while t can fail in the version of $c-1$ (i.e., the commit before c). Then, we integrate the creation of the test case t into the code commit history[1]. Specifically, we create two branches, i.e., "fix" branch from the commit c and "buggy" branch from the commit $c-1$. Then, we introduce a commit for adding the test case t based on both branches. By this means, we can check out a buggy version and a fixed version from the Git repository.

[1]Considering the extensibility of *gDefects4DL* to facilitate more software engineering tasks in the future, we backup the whole Git repository of the defect.

Table 1: Types of general bugs of deep learning programs, and their support by three state-of-the-art tools. Some bug types have not been supported by any tools, the corresponding entry is marked as "/". Some tools cannot support some framework such as Keras or PyTorch, thus the corresponding entry is the number less than the total number of bugs under the same type.

Type	Description	Bug Number	Tools		
			ShapeFlow	DEBAR	GRIST
Shape Mismatch	The tensors are operated (e.g., multiplied) with incorrect dimensionality. In gDefects4dl, we focus on the common patterns of their fix solutions.	11	7	/	/
API Misuse	Mistakes made when invoking a library call. In gDefects4dl, we focus on the common root causes or mistakes leading to the general bugs.	23	/	/	/
Numeric Error	The NaN value caused by some operation such as zero division error. In gDefects4dl, we focus on the common patterns of their fix solutions.	8	/	4	6
Type Mismatch	The deep-learning specific type mismatch, e.g., The tensor type is misused as numpy type, or numpy-float type is misused as numpy-int type.	16	/	/	/
Architecture Convention Violation	The architecture convention is violated, e.g., a normalization layer such as softmax or sigmoid needs to applied as the last layer of a classifier.	4	/	/	/
Performance Bug	The performance bugs with fixed pattern, e.g., improve the training performance if a machine is equipped with multiple GPUs.	2	/	/	/

Moreover, we manually label (i.e., isolate) which changes are the fix to an individual bug.

Environment Isolation. The bugs can only be replicated with specific environments, i.e. specific dataset, depending libraries of specific versions, and specific configuration of operation systems. Therefore, after we enhance the test case, we provide two ways to isolate the environment.

- **Conda Version:** In this version, we provide a conda environment for each bug, where we prepare the scripts of downloading dataset and depending libraries (including the integrated tools). Users need to run the scripts in conda to set up the environment before replicating the bugs.
- **Docker Version:** In this version, we pre-install all the prerequisite environments of each bug in an individual Docker image. Users can replicate the bug in Docker container.

The conda version is designed for the sake of maintaining *gDefects4DL* with a small size of disk. In contrast, the docker version is designed for the convenience of installation and the future maintenance of bugs related to system configuration.

Meta Information Extraction and Tool Integration. We manually attach the meta information for each bug, such as bug id, summarized bug type, issue URL, defect description, support level, and the reason for why the bug is general. In addition, we integrate three tools (ShapeFlow [12], DEBAR [19] and GRIST [15]) to support their evaluation on the general bugs.

3 TOOL DESIGN

Figure 2 shows our design of *gDefects4DL* dataset, consisting of user interface layers, defect abstraction layer, and isolated environment layer. As mentioned above, for each bug, we provide both docker container and conda-setup script to provide an isolated environment for its replication. In either environment, we also support its version control system (via git) and detection tools (e.g., ShapeFlow, GRIST, and DEBAR). In the defect abstraction layer, we provide a set of programming interfaces for downloading a bug, finding/viewing certain bugs, replicating a specific bug, and running detection

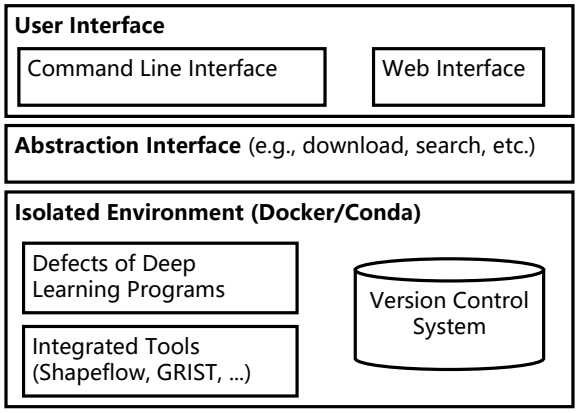

Figure 2: Overview of *gDefects4DL* Design

tools for a bug. In the user interface, we provide both command line interface and user interface. More details can be checked at our demonstrated video and website.

4 USER INTERFACE FOR FEATURES

All the features are supported in both command line and web frontend. Readers can check our user manual/videos for more details.

4.1 Web Frontend

The web frontend is designed for users to browse the overall information of each recorded bug. In *gDefects4DL*, users can search bugs based on the bug type (e.g., API misuse, shape-mismatch, etc.), framework (e.g., Tensorflow or PyTorch), error message, supported tools (e.g., DEBAR or ShapeFlow). Users can install the bug either by setting up the conda environment or pulling its corresponding docker image (from DockerHub).

As shown in Figure 3, for each bug, we further show its detailed information including bug id, bug type, support level, error message, and description in the "bug details" webpage. *gDefects4DL*

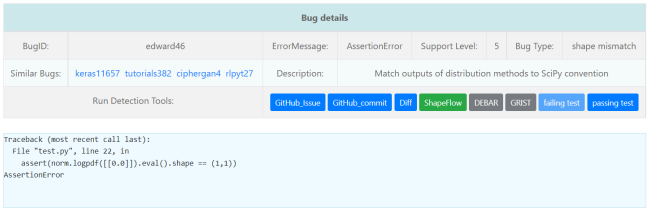

Figure 3: A screenshot of the web frontend of *gDefects4DL* to view an individual bug

further presents which Github issue reports the bug (along with its discussion), and its fixing commit. Clicking the corresponding button allows the user to visit their corresponding URL on Github. The "diff" button further shows the fixing change in the fixing commit. Next, user can click "passing test" or "failing test" to view the error message (on the web console) produced by running the test on the fixed and buggy revision. Moreover, in the region of "detection tools", we show three tools integrated in *gDefects4DL*. The grey button indicates that the bug is not supported by the tool, in contrast, the green button indicates the tool is runnable against the bug. Note that, some tools like DEBAR only support Tensorflow framework, which cannot infer the deep learning program bugs in PyTorch framework. The running message will be shown on the web console. Finally, for each general bug, we will also show the other bugs sharing similar root cause and fix solution, as shown in the region of "similar bugs". Clicking each bug id allows user to view its own "bug detail" page.

4.2 Command Line End

The command line end is equipped with the same functionalities as provided by the web frontend. Each function specified in Section 4.1 is wrapped as an API for user to invoke. The Java Doc of *gDefects4DL* API is available at http://47.93.14.147:9000/javadoc. In addition, the conda and docker environment allows the user to investigate the source code of each project and their evolution history.

5 DISCUSSION

In this section, we discuss the limitations and future work of *gDefects4DL*, which lies in two folds.

Scalability. *gDefects4DL* is still a relatively small dataset consisting of 64 bugs. In this work, we develop a semi-automated workflow (as Figure 1) to save manual efforts. Automated solutions to construct Java bug dataset have emerged, such as RegMiner [10] and BugSwarm [11]. We foresee a more automated solution to improve the scalability of dataset construction.

Diversity. We select the general bugs with regard to their diversity. Nevertheless, there are still general bug types which have not been included in *gDefects4DL*. For example, a bug can be caused by noisy training dataset, or lies in the deep learning framework like PyTorch and Tensorflow. Such types have not been included for now. In our future work, we will further improve the diversity of *gDefects4DL* for facilitating software engineering research on deep learning program.

6 RELATED WORK

6.1 Deep Learning Bug Datasets

Existing researchers have constructed a number of datasets of deep learning bugs. Zhang et al. pioneered the collection of deep learning programs from both Stackoverflow and Github, followed with an empirical study to categorize bug root cause and symptoms [18]. Following their work, Zhang et al. enlarge the scalability of the empirical study by collecting failures of deep learning jobs from Microsoft [17]. Humbatova et al. collect deep learning bugs and propose a more sophisticated taxonomy with a survey supported by 21 developers [2]. In the meantime, Islam et al. propose a taxonomy of deep learning program bugs [3] and their fix patterns [4].

gDefects4DL is different from those works in two folds. First, *gDefects4DL* focuses on the bugs with much higher potential to be generalized to other projects. We have shown examples in Listing 1 and Listing 2. Each bug is also described with a support level, indicating how many bugs sharing the similar root cause and fix solution. We believe that, for developing automated software engineering techniques, a dataset of bugs with higher generality is preferred. Second, *gDefects4DL* targets for the general bugs in sophisticated projects, and captures more comprehensive bug information such as bug evolution history, the environment of replication, and even the description for their root cause and generality.

6.2 Deep Learning Fault Localization

The datasets of deep learning programs give rise to a number of techniques to locate and repair the bugs. Verma et al. propose ShapeFlow [12] to detect potential shape-mismatch problem of deep learning programs. Zhang et al. propose DEBAR [19] by statically detecting numeric bugs with an abstraction interpretation technique. Yan et al. propose GRIST [15] which improves DEBAR by inferring the location of numeric errors via gradient back-propagation. Mohammad et al. propose DeepLocalize [14] to capture numeric errors such as disappearing and exploding gradients.

We integrate the most of the above tools in *gDefects4DL*, which (1) confirms their effectiveness and (2) presents the engineering effort requires to improve their generality and the need of new tools to localize and repair new types of bugs.

7 CONCLUSION AND FUTURE WORK

In this work, we introduce *gDefects4DL* which is the first general deep learning bug dataset, aiming to facilitate more researches of deep learning programs in the community. *gDefects4DL* consists of 64 general bugs, which isolates the runtime environment via both conda and docker settings, code evolution history, and the integration of three tools. In the future, we will extend *gDefects4DL* to support more general deep learning program bugs, visualized UI for users, incorporate more baseline approaches such as DeepVisualInsight [16] and DeepLocalize [14], and even develop our own static and dynamic bug detection and analysis techniques [5–9, 13], to facilitate controlled research experiments.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their valuable input to improve our work. This work was supported in part by the Minister of Education, Singapore (No. MOET32020-0004, T2EP20120-0019, and T1-251RES1901), the National Research Foundation Singapore through its National Satellite of Excellence in Trustworthy Software Systems (NSOE-TSS) office (Award Number: NSOE-TSS2019-05), and the National Natural Science Foundation of China under Grant No. 61832014.

REFERENCES

[1] Qianyu Guo, Sen Chen, Xiaofei Xie, Lei Ma, Qiang Hu, Hongtao Liu, Yang Liu, Jianjun Zhao, and Xiaohong Li. 2019. An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 810–822.

[2] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1110–1121.

[3] Md Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. 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*. 510–520.

[4] Md Johirul Islam, Rangeet Pan, Giang Nguyen, and Hridesh Rajan. 2020. Repairing deep neural networks: Fix patterns and challenges. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1135–1146.

[5] Yun Lin, Guozhu Meng, Yinxing Xue, Zhenchang Xing, Jun Sun, Xin Peng, Yang Liu, Wenyun Zhao, and Jinsong Dong. 2017. Mining implicit design templates for actionable code reuse. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 394–404.

[6] Yun Lin, You Sheng Ong, Jun Sun, Gordon Fraser, and Jin Song Dong. 2021. Graph-based seed object synthesis for search-based unit testing. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1068–1080.

[7] Yun Lin, Jun Sun, Gordon Fraser, Ziheng Xiu, Ting Liu, and Jin Song Dong. 2020. Recovering fitness gradients for interprocedural Boolean flags in search-based testing. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 440–451.

[8] Yun Lin, Jun Sun, Lyly Tran, Guangdong Bai, Haijun Wang, and Jinsong Dong. 2018. Break the dead end of dynamic slicing: Localizing data and control omission bug. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. 509–519.

[9] Yun Lin, Jun Sun, Yinxing Xue, Yang Liu, and Jinsong Dong. 2017. Feedback-based debugging. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 393–403.

[10] Xuezhi Song, Yun Lin, Siang Hwee Ng, Ping Yu, Xin Peng, and Jin Song Dong. 2021. Constructing Regression Dataset from Code Evolution History. *arXiv preprint arXiv:2109.12389* (2021).

[11] David A Tomassi, Naji Dmeiri, Yichen Wang, Antara Bhowmick, Yen-Chuan Liu, Premkumar T Devanbu, Bogdan Vasilescu, and Cindy Rubio-González. 2019. Bugswarm: Mining and continuously growing a dataset of reproducible failures and fixes. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 339–349.

[12] Sahil Verma and Zhendong Su. 2020. ShapeFlow: Dynamic Shape Interpreter for TensorFlow. *arXiv preprint arXiv:2011.13452* (2020).

[13] Haijun Wang, Yun Lin, Zijiang Yang, Jun Sun, Yang Liu, Jin Song Dong, Qinghua Zheng, and Ting Liu. 2019. Explaining regressions via alignment slicing and mending. *IEEE Transactions on Software Engineering* (2019).

[14] Mohammad Wardat, Wei Le, and Hridesh Rajan. 2021. DeepLocalize: Fault Localization for Deep Neural Networks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 251–262.

[15] Ming Yan, Junjie Chen, Xiangyu Zhang, Lin Tan, Gan Wang, and Zan Wang. 2021. Exposing numerical bugs in deep learning via gradient back-propagation. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 627–638.

[16] Xianglin Yang, Yun Lin, Ruofan Liu, Zhenfeng He, Chao Wang, Jin Song Dong, and Hong Mei. 2022. DeepVisualInsight: Time-Travelling Visualization for Spatio-Temporal Causality of Deep Classification Training. *The Thirty-Sixth AAAI Conference on Artificial Intelligence* (2022).

[17] Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. 2020. An empirical study on program failures of deep learning jobs. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1159–1170.

[18] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An empirical study on TensorFlow program bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 129–140.

[19] Yuhao Zhang, Luyao Ren, Liqian Chen, Yingfei Xiong, Shing-Chi Cheung, and Tao Xie. 2020. Detecting numerical bugs in neural network architectures. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 826–837.