# Impact of Programming Languages on Machine Learning Bugs

Sebastian Sztwiertnia*
Maximilian Grübel*
Amine Chouchane*
Technical University of Darmstadt
Germany
firstname.lastname@stud.tu-darmstadt.de

Daniel Sokolowski
Krishna Narasimhan
Mira Mezini
Technical University of Darmstadt
Germany
{sokolowski,kri.nara,mezini}@cs.tu-darmstadt.de

## ABSTRACT

Machine learning (ML) is on the rise to be ubiquitous in modern software. Still, its use is challenging for software developers. So far, research has focused on the ML libraries to find and mitigate these challenges. However, there is initial evidence that programming languages also add to the challenges, identifiable in different distributions of bugs in ML programs. To fill this research gap, we propose the first empirical study on the impact of programming languages on bugs in ML programs. We plan to analyze software from GitHub and related discussions in GitHub issues and Stack Overflow for bug distributions in ML programs, aiming to identify correlations with the chosen programming language, its features and the application domain. This study's results enable better-targeted use of available programming language technology in ML programs, preventing bugs, reducing errors and speeding up development.

## CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Software and its engineering** → **General programming languages**; • **General and reference** → **Empirical studies**.

## KEYWORDS

machine learning, programming languages, empirical study

**ACM Reference Format:**
Sebastian Sztwiertnia, Maximilian Grübel, Amine Chouchane, Daniel Sokolowski, Krishna Narasimhan, and Mira Mezini. 2021. Impact of Programming Languages on Machine Learning Bugs. In *Proceedings of the 1st ACM International Workshop on AI and Software Testing/Analysis (AISTA '21), July 12, 2021, Virtual, Denmark*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3464968.3468408

## 1 INTRODUCTION AND BACKGROUND

The popularity and relevance of ML steadily increases. This ubiquity makes code quality and bug prevention in ML programs more relevant than ever. Bugs increase development costs tremendously and can lead to severe accidents. Between 40% to 80% of the project cost can be attributed to software testing [7]. Fixing ML bugs takes a lot of time; *timing and optimization* bugs on average 72 days and *algorithm and method* bugs—most common in ML—require on average 92 days [29]. Unidentified bugs may lead to reduced model performance [13], and they can cause real-world accidents, e.g., the Soyuz TMA-1 spaceship missed its supposed landing spot [23]. Also, 60% of the bugs analyzed by Islam et al. [13] lead to an application crash, enforcing the importance of preventing them. ML also becomes popular in safety-critical applications, e.g., autonomous driving [31], making its correctness more relevant than ever.

Existing research on bugs in ML programs focuses on ML libraries and investigates the developers' usage and problems [13, 28, 29, 32]. However, not only libraries but also the used programming language can impact code quality and encountered bug characteristics. Islam et al. [13] provide first evidence by analyzed bug distributions in ML programs. They found similar bug characteristics across the investigated ML libraries, hinting that there are common problems that might be caused by the language. Yet, so far, there is no focused investigation of the impact of the programming language nor its features on bugs in ML programs.

Today Python is central to ML development, especially for model development and integration. According to a survey on the popular ML platform Kaggle [15], Python is the most often used and the most recommended language to aspiring data scientists; presumably due to its perceived user-friendliness. However, Python lacks in speed compared to other languages, has higher memory consumption, and is more error-prone due to its dynamic typing, which can be frustrating [21]. It may well be that a considerable number of bugs are due to the prevalence of Python. But, we simply do not know.

For general purpose software on GitHub repositories, studies investigated the impact of programming languages on code quality [3, 24]. While these studies come to different findings regarding a programming language's impact, they underline the complexity of this question, where already finding suitable metrics is challenging [10]. Further, ML programs are different, making it hard to apply engineering know-how from general software programs or utilizing standard static analyzers to find bugs. The ML development process is data-driven in contrast to traditional code-driven software development [1] and requires closer collaboration between separate professions, including domain experts, data scientists, software engineers, operators and more [19].

In this paper, we aim to fill this gap by designing and conducting a systematic analysis of the effect of the chosen different programming language on bugs in ML programs. Gaining insight into whether and how programming languages influence bugs in ML programs will steer language development towards preventing ML bugs. Further, it can improve the language selection, which reduces the probability to face bugs, speeding up development and saving

*The first three authors contributed equally to this research.


*AISTA '21, July 12, 2021, Virtual, Denmark*

ACM ISBN 978-1-4503-8541-1/21/07…$15.00
https://doi.org/10.1145/3464968.3468408

cost. Also, the ML community can benefit as a whole by receiving tools and features better tailored to their needs.

Therefore, in this paper:

- We propose an empirical study on bugs in ML programs on GitHub and Stack Overflow (§2), correlating bug distributions with (1) the chosen programming language, (2) the application domain, and (3) the programming language's features.
- We present initial evidence for our research objectives (§3).
- We analyze the potential impact of our study, enabling better application of programming languages to ML programs (§4).

## 2 INVESTIGATING THE PROGRAMMING LANGUAGE IMPACT ON ML BUGS

To shed light on the impact of programming languages on bug characteristics in ML programs, we propose to study existing ML programs on GitHub and related problem discussions in GitHub issues and Stack Overflow questions and answers. We first ask:

**RQ1: Do the bug characteristics depend on the chosen programming language?** We assess whether the problems and peculiarities in the data can be attributed to the chosen programming language. We expect that the bug characteristics of ML programs can be grouped into significantly different clusters, which are mapped to their programming languages. Then we ask:

**RQ2: Does the application domain influence the bug characteristics within a chosen programming language?** To achieve guidance in choosing the right language, we aim to identify whether the application domain, e.g., computer vision or email filtering, correlates with the bug characteristics. In case, we provide evidence that future research on languages for ML must consider the application domain. Otherwise, likely, language improvements for one domain generalize well to other domains. To further detail, we ask:

**RQ3: Are differences in the bug distribution explainable by the features of the chosen programming language?** We assess bugs present in one language but unlikely in another and analyze correlations with the languages' features. This includes finding misuses of features and identifying the lacking features to prevent bugs. As Python is the most common ML language, we especially expect to find bugs that could be prevented by features available in other languages that are absent in Python.

### 2.1 Impact of the Language Choice

To answer RQ1, we first collect a sufficiently big dataset from Stack Overflow and GitHub, common sources for such studies [3, 11, 13, 24, 33]. Stack Overflow is a questions and answers platform for developers seeking help with programming problems. Its tagging system assigns a programming language to each question, making it easy to extract data. GitHub is a hosting platform for code repositories. Like Stack Overflow, GitHub has a tagging system for programming languages and a public API to download the data.

First, we download all GitHub repositories by using the same methodology as Gonzalez et al. [11] as it resulted in a sufficient amount of repositories (4.524) and excludes irrelevant repositories such as tutorials, homework assignments, etc. Stack Overflow questions and answers related to ML projects are gathered using the methodology of Islam et al. [13]. In this study, we focus on Python, C++, JavaScript, Java, R and Go as these are the main programming languages represented in the data of Gonzalez et al. [11]. We are aware of programming languages offering ML capabilities inherently, e.g., Julia and Swift. Due to these special capabilities, we suspect an impact on bug distributions that is too unique to compare to the prior mentioned general-purpose languages. Therefore, we do not include Swift and Julia in our proposed study.

In the next step, we will apply the pre-processing procedure proposed by Islam et al. [13], filtering and labeling the data. We reduce the data to bug-related content by selecting only the GitHub commits whose message includes "fix". For GitHub issues and Stack Overflow questions and answers, we will filter by keywords, too, extracting all posts containing "bug", "error" or "fail". For our study, the bug type, bug effect, root cause and programming language are the necessary attributes that we will extract. To categorize bugs, we will use the taxonomy of Beizer [2], also used by Islam et al. [13].

In the analysis phase, we identify the distributions of bug characteristics for each programming language. To compare the discrete distributions pairwise, we will use the Kolmogorov-Smirnov test (KS test) [18]. It prevents the unjustified assumption of normal distribution and comparing means, in contrast to the t-test statistic used by Islam et al. [13]. We chose the significance level $\alpha = 5\%$.

For instance, for the characteristic *bug type*, we will find the language pairs that have *significantly* different occurrence of these bugs. Finding these differences would support our hypothesis that the choice of the programming language impacts the correctness of ML programs. Also, the findings can indicate which languages are better suited for the development of ML-related applications as their choice is likely to reduce the number of bugs.

### 2.2 Impact of the Application Domain

To answer RQ2, we reuse the dataset collected in §2.1. However, for the analysis of the application domain's impact, we add the application domain attribute by tagging the data accordingly. Based on Shinde and Shah [27] we will categorize into the five domains *Computer Vision*, *Prediction*, *Semantic Analysis*, *Natural Language Processing* and *Information Retrieval*.

The analysis will follow the same procedure as in §2.1, however, we now test the impact of the application domain on the observed bug characteristics distribution instead of the programming language. We apply the KS test to each pair of domains across all programming languages and within each language. This obtains the domain pairs with *significantly* different bug characteristics across all languages and separately for each programming language.

If we observe that bug distributions are significantly different between domains across languages, we show that programming language research for ML must take the application domains into account. If the insights differ for the programming languages, we find the languages that are least and most error-prone for each investigated domain. This would illustrate that an ML project's language selection should take the application domain into account and provides qualified guidance during this process.

### 2.3 Impact of Programming Language Features

To answer RQ3, we reuse the dataset from §2.1 and combine it with a dataset of programming language features available in each of the investigated languages. Based on [8], we propose using the taxonomy introduced by Jordan et al. [14] with the addition of the class of programming language. This taxonomy includes *type checking*,

*state cell assignment*, *state cell deletion*, *high order types*, *single assignment*, *modularity unit*, *functions* and *interactive input/output*. These features are complemented, as described, by the differentiation of programming language classes: *scripting language*, *object-oriented language* and *functional language* based on Scott [25]. To distinguish between language classes is reasonable as it was found that certain language classes are less prone to error than others [24].

In the analysis, we reuse the method used in §2.1, but this time, we assess for each language feature separately the impact of its availability on the bug characteristics distribution. We will find the language features whose availability correlates with the bugs in ML programs. For these, we repeat the KS test for each bug category individually and apply further manual analysis, finding which bugs are prevented by the feature and explanations why.

With the knowledge gained through this study, a set of language features could be found that potentially minimizes bugs in ML programs. This can be used to guide language selection and to shape future language and ML library development, suggesting to support language features that are likely to prevent bugs.

## 3 INITIAL EVIDENCE

*ML Programs Are Different.* The development of ML-driven applications does not follow the traditional software development workflow and strongly focuses on data [1]. Islam et al. [13] show that data bugs are most common in ML programs, further supporting that ML faces new problems that do not exist in traditional software development. Additionally, current debugging practices do not support identifying bugs in ML code well [32], indicating differences in the bugs and their distribution between ML and non-ML programs. Humbatova et al. [12] analyzed projects using popular deep learning frameworks, derive a bug taxonomy and validate it with a user study. While they did not investigate the impact of the used programming language, they give evidence to the relevance and difference of ML bugs. Also, the development of ML programs is different. Traditionally, the software design is secured from unsolicited change, hindering ML data scientists in their exploration [26]. Data scientists today also rely on informal versioning, consisting of commenting out parts of the code to keep it for later reference [16]. This approach of exploratory programming leads to new code quality trade-offs [4]. These insights show that ML programs are different from programs discussed in software engineering so far and, thus, provide evidence that this ML-focused study is needed.

*Programming Languages Impact Bug Characteristics.* Various studies analyzed code quality based on the programming language choice for general programs by, e.g., relating the language choice to the programs' bug distribution. Ray et al. [24] found a correlation between the chosen language and software bug distributions. However, Berger et al. [3] could only partly reproduce these results. Programming languages also impact bugs in projects utilizing multiple languages. Kochhar et al. [17] showed that the degree of error-proneness changes when specific programming languages are added to a project. In particular, the error-proneness increased when adding C++, Objective-C, Java, TypeScript, Clojure or Scala. This indicates that the languages may have different bug characteristics, providing initial evidence to RQ1. However, detailed insight for ML-related programs is not available yet.

*Application Domains Impact Bug Characteristics.* Linares-Vásquez et al. [20] analyzed the code quality of Java projects for different application domains. They used object-oriented metrics [6] as a proxy for code quality and found a negative correlation between anti-patterns and object-oriented metrics. Linares-Vásquez et al. [20] found that anti-patterns varied across application domains. Further, Islam et al. [13] identified that bug patterns in ML programs are different compared to other software domains. This provides initial evidence to RQ2, showing for general programs that their bug characteristics depend on the application domain and that ML programs have different characteristics than non-ML ones. However, a detailed analysis for application domains within ML is missing yet.

*Programming Language Features Prevent Bugs.* Programming languages differ in their set of supported features. Some of these can reduce the amount of bugs as shown, e.g., for type systems [22]. In contrast, dynamically typed languages like Python detect these errors at run time, giving less guarantee that a program will work. E.g., Islam et al. [13] confirm by identifying that Python is prone to data bugs and that type and shape mismatches are common problems. Programming language research is focused on developing language features preventing the introduction of bugs. For instance, gradual type systems have been explored recently for dynamically typed languages, including Python [30] and JavaScript [5]. E.g., TypeScript is a superset of JavaScript with static type system features and Gao et al. [9] observed that it could have prevented 15% of bugs in the investigated code on GitHub. These efforts to prevent common errors by adding a feature to the language give initial evidence to RQ3. However, the relationship between present errors in ML programs and language features is yet to be discovered.

## 4 IMPACT ANALYSIS

The presented study potentially impacts all future ML programs and, due to ML's ubiquity, a huge share of future systems. The improved understanding of the languages' impact and flaws for ML programs can help to select better suiting ones for the faced problem as well as improving the languages themselves. Both potentially improve ML program correctness and eases development, which can speed up ML development and further propel its ubiquity.

In detail, the insights through RQ1, RQ2 and RQ3 can help already today to identify preferable languages for ML problems from a bug prevention perspective. RQ2 guides programming language researchers for ML languages on how central the application domain is for their contributions, i.e., how probable it is that their evaluated improvements generalize to other domains. Further, RQ2 and especially RQ3 are valuable sources to guide the development of programming languages and libraries for future ML programs. ML library authors can leverage the identified shortcomings and apply language-level solutions as their library APIs themselves usually can be seen as a language within the programming language (an embedded DSL). Even without altering their APIs, library authors can improve their documentation and training through the improved awareness of mishaps not prevented by the language.

Reducing the number of introduced bugs also reduces friction and frustration during the development of ML programs. Thus, based on RQ2 and RQ3, easier-to-learn languages may be selected for teaching. This helps the community to grow fast and shapes the shared knowledge and standards within the community.

## 5 CONCLUSION

In this paper, we propose an empirical study on programming languages and PL bugs based on data from Stack Overflow and GitHub. First, we assess the impact of the programming language choice on bug distributions in ML programs. Second, we investigate the effect application domains have on language-specific and general bug distributions in ML programs. Lastly, we examine the programming language features' impact. We outline the method and research plan and identify initial evidence. The results will impact practitioners and scientists, revealing the code quality effects of programming languages in ML, potentially guiding language choice and feature design, preventing bugs in future ML programs.

## ACKNOWLEDGMENTS

This work has been co-funded by the German Research Foundation (SFB 1119), by the Hessian LOEWE initiative (Software-Factory 4.0), and by the German Federal Ministry of Education and Research (BMBF) and the Hessian Ministry of Higher Education, Research and the Arts (HMKW) within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

## REFERENCES

[1] Saleema Amershi, Andrew Begel, Christian Bird, et al. 2019. Software Engineering for Machine Learning: A Case Study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (Montreal, Quebec, Canada) *(ICSE-SEIP '19)*. IEEE Press, 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[2] Boris Beizer. 1984. *Software System Testing and Quality Assurance.* Van Nostrand Reinhold Co., USA.

[3] Emery D. Berger, Celeste Hollenbeck, Petr Maj, Olga Vitek, and Jan Vitek. 2019. On the Impact of Programming Languages on Code Quality: A Reproduction Study. *ACM Trans. Program. Lang. Syst.* 41, 4, Article 21 (Oct. 2019), 24 pages. https://doi.org/10.1145/3340571

[4] Mary Beth Kery and Brad A. Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 25–29. https://doi.org/10.1109/VLHCC.2017.8103446

[5] Gavin Bierman, Martín Abadi, and Mads Torgersen. 2014. Understanding TypeScript. In *Proceedings of the 28th European Conference on ECOOP 2014 — Object-Oriented Programming - Volume 8586*. Springer-Verlag, Berlin, Heidelberg, 257–281. https://doi.org/10.1007/978-3-662-44202-9_11

[6] S.R. Chidamber and C.F. Kemerer. 1994. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* 20, 6 (1994), 476–493. https://doi.org/10.1109/32.295895

[7] S. Eldh, H. Hansson, S. Punnekkat, et al. 2006. A Framework for Comparing Efficiency, Effectiveness and Applicability of Software Testing Techniques. In *Testing: Academic Industrial Conference - Practice And Research Techniques (TAIC PART'06)*. 159–170. https://doi.org/10.1109/TAIC-PART.2006.1

[8] Onyeka Ezenwoye. 2018. What Language? - The Choice of an Introductory Programming Language. In *2018 IEEE Frontiers in Education Conference (FIE)*. 1–8. https://doi.org/10.1109/FIE.2018.8658592

[9] Zheng Gao, Christian Bird, and Earl T. Barr. 2017. To Type or Not to Type: Quantifying Detectable Bugs in JavaScript. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 758–769. https://doi.org/10.1109/ICSE.2017.75

[10] M. Garkavtsev, N. Lamonova, and A. Gostev. 2018. Chosing a Programming Language for a New Project from a Code Quality Perspective. In *2018 IEEE Second International Conference on Data Stream Mining Processing (DSMP)*. 75–78. https://doi.org/10.1109/DSMP.2018.8478454

[11] Danielle Gonzalez, Thomas Zimmermann, and Nachiappan Nagappan. 2020. The State of the ML-Universe: 10 Years of Artificial Intelligence & Machine Learning Software Development on GitHub. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 431–442. https://doi.org/10.1145/3379597.3387473

[12] 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* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1110–1121. https://doi.org/10.1145/3377811.3380395

[13] 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* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 510–520. https://doi.org/10.1145/3338906.3338955

[14] Howell Jordan, Goetz Botterweck, John Noll, et al. 2015. A feature model of actor, agent, functional, object, and procedural programming languages. *Science of Computer Programming* 98 (2015), 120–139. https://doi.org/10.1016/j.scico.2014.02.009

[15] Kaggle. 2019. State of Data Science and Machine Learning 2019. https://www.kaggle.com/c/kaggle-survey-2019/data, last accessed on 2021-06-03.

[16] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1265–1276. https://doi.org/10.1145/3025453.3025626

[17] P. S. Kochhar, D. Wijedasa, and D. Lo. 2016. A Large Scale Study of Multiple Programming Languages and Code Quality. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 563–573. https://doi.org/10.1109/SANER.2016.112

[18] Andrey Kolmogorov. 1933. Sulla determinazione empirica di una lgge di distribuzione. *Inst. Ital. Attuari, Giorn.* 4 (1933), 83–91.

[19] Grace A. Lewis, Stephany Bellomo, and Ipek Ozkaya. 2021. Characterizing and Detecting Mismatch in Machine-Learning-Enabled Systems. arXiv:2103.14101 https://arxiv.org/abs/2103.14101

[20] Mario Linares-Vásquez, Sam Klock, Collin McMillan, et al. 2014. Domain Matters: Bringing Further Evidence of the Relationships among Anti-Patterns, Application Domains, and Quality-Related Metrics in Java Mobile Apps. In *Proceedings of the 22nd International Conference on Program Comprehension* (Hyderabad, India) *(ICPC 2014)*. Association for Computing Machinery, New York, NY, USA, 232–243. https://doi.org/10.1145/2597008.2597144

[21] Abhinav Nagpal and Goldie Gabrani. 2019. Python for Data Analytics, Scientific and Technical Applications. In *2019 Amity International Conference on Artificial Intelligence (AICAI)*. 140–145. https://doi.org/10.1109/AICAI.2019.8701341

[22] S. Nanz and C. A. Furia. 2015. A Comparative Study of Programming Languages in Rosetta Code. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 778–788. https://doi.org/10.1109/ICSE.2015.90

[23] D.L. Parnas and M. Lawford. 2003. The role of inspection in software quality assurance. *IEEE Transactions on Software Engineering* 29, 8 (2003), 674–676. https://doi.org/10.1109/TSE.2003.1223642

[24] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A Large Scale Study of Programming Languages and Code Quality in Github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 155–165. https://doi.org/10.1145/2635868.2635922

[25] Michael L. Scott. 2009. 1 - Introduction. In *Programming Language Pragmatics (Third Edition)* (third edition ed.), Michael L. Scott (Ed.). Morgan Kaufmann, Boston, 5–39. https://doi.org/10.1016/B978-0-12-374514-9.00010-0

[26] Beau Sheil. 1986. Datamation®: Power Tools for Programmers. In *Readings in Artificial Intelligence and Software Engineering*, Charles Rich and Richard C. Waters (Eds.). Morgan Kaufmann, 573–580. https://doi.org/10.1016/B978-0-934613-12-5.50048-3

[27] Pramila P. Shinde and Seema Shah. 2018. A Review of Machine Learning and Deep Learning Applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. 1–6. https://doi.org/10.1109/ICCUBEA.2018.8697857

[28] X. Sun, T. Zhou, G. Li, J. Hu, H. Yang, and B. Li. 2017. An Empirical Study on Real Bugs for Machine Learning Programs. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. 348–357. https://doi.org/10.1109/APSEC.2017.41

[29] F. Thung, S. Wang, D. Lo, and L. Jiang. 2012. An Empirical Study of Bugs in Machine Learning Systems. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*. 271–280. https://doi.org/10.1109/ISSRE.2012.22

[30] Michael M. Vitousek, Andrew M. Kent, Jeremy G. Siek, and Jim Baker. 2014. Design and Evaluation of Gradual Typing for Python. *SIGPLAN Not.* 50, 2 (Oct. 2014), 45–56. https://doi.org/10.1145/2775052.2661101

[31] Jianxiong Xiao. 2017. Learning Affordance for Autonomous Driving. In *Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services* (Snowbird, Utah, USA). Association for Computing Machinery, New York, NY, USA, 1. https://doi.org/10.1145/3131944.3133941

[32] Ru Zhang, Wencong Xiao, Hongyu Zhang, et al. 2020. An Empirical Study on Program Failures of Deep Learning Jobs. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. 1159–1170. https://doi.org/10.1145/3377811.3380362

[33] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, et al. 2018. Are Code Examples on an Online Q A Forum Reliable?: A Study of API Misuse on Stack Overflow. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 886–896. https://doi.org/10.1145/3180155.3180260