Secure Deep Learning Engineering: a Road towards Quality Assurance of Intelligent Systems

Yang Liu[1], Lei Ma[2], and Jianjun Zhao[2]

[1] Nanyang Technological University, Singapore
[2] Kyushu University, Japan

Abstract. Over the past decades, deep learning (DL) systems have achieved tremendous success and gained great popularity in various applications, such as intelligent machines, image processing, speech processing, and medical diagnostics. Deep neural networks are the key driving force behind its recent success, but still seem to be a magic black box lacking interpretability and understanding. This brings up many open safety and security issues with enormous and urgent demands on rigorous methodologies and engineering practice for quality enhancement. A plethora of studies have shown that state-of-the-art DL systems suffer from defects and vulnerabilities that can lead to severe loss and tragedies, especially when applied to real-world safety-critical applications.
In this paper, we perform a large-scale study and construct a paper repository of 223 relevant works to the quality assurance, security, and interpretation of deep learning. Based on this, we, from a software quality assurance perspective, pinpoint challenges and future opportunities to facilitate drawing the attention of the software engineering community towards addressing the pressing industrial demand of secure intelligent systems.

Keywords: Artificial intelligence · Deep learning · Software engineering · Security · Quality Assurance · Reliability · Deep Learning Engineering

1 Introduction

In company with massive data explosion and powerful computational hardware enhancement, deep learning (DL) has recently achieved substantial strides in cutting-edge intelligent applications, ranging from virtual assistant (e.g., Alex, Siri), art design [18], autonomous vehicles [13, 19], to medical diagnoses [1, 3] – tasks that until a few years ago could be done only by humans. DL has become the innovation driving force of many next generation's technologies. We have been witnessing on the increasing trend of industry stakeholders' continuous investment on DL based intelligent system [5–8, 40], penetrating almost every application domain, revolutionizing industry manufacturing as well as reshaping our daily life.

However, current DL system development still lacks systematic engineering guidance, quality assurance standards, as well as mature toolchain support. The *magic box*, such as DL training procedure and logic encoding (as high dimensional weight matrices and complex neural network structures), further poses challenges to interpret and understand behaviors of derived DL systems [4, 16, 26]. The latent software quality

and security issues of current DL systems, already started emerging out as the major vendors, rush in pushing products with higher intelligence (e.g., Google/Uber car accident [21, 41], Alexa and Siri could be manipulated with hidden command [39]. A DL image classifier with high test accuracy is easily fooled by a single-pixel perturbation [2]). Deploying such cocooned DL systems to the real-world environment without quality and security assurance leaves high risks, where newly evolved cyber- and adversarial-attacks are inevitable.

To bridge the pressing industry demand and future research directions, this paper first performs a large-scale empirical study on the most-recent curated 223 relevant works on deep learning engineering from a software quality assurance perspective. Based on this, we perform a quantitative and qualitative analysis to identify the common issues that the current research community most dedicated to. With an in-depth investigation on current works, and our in-company DL development experience obtained, we find that the development of secure and high quality deep learning systems requires enormous engineering effort, while most AI communities focus on the theoretical or algorithmic perspective of deep learning. Indeed, the development of modern complex deep learning systematic solutions could be a challenge for an individual research community alone. We propose the *Secure Deep Learning Engineering* (SDLE) development process specialized for DL software, which we believe is an interdisciplinary future direction (e.g., AI, SE, security) towards constructing DL applications, in a systematic method from theoretical foundations, software & system engineering, to security guarantees. We further discuss current challenges and opportunities in SDLE from a software quality assurance perspective.

To the best of our knowledge, our work is the first study to vision SDLE, from the quality assurance perspective, accompanied by a state-of-the-art literature curation. We hope this work facilitates drawing the attention of the software engineering community on necessity and demands of quality assurance for SDLE, which altogether lays down the foundations and conquers technical barriers towards constructing robust and high-quality DL systems. The repository is available at: `https://sdle2018.github.io/`

2 Research Methodology

This section summarizes our concerned research questions, and discusses the detail of paper collection procedure for further analysis.

2.1 Research Questions

This paper mainly focuses on following research questions.

- **RQ-1:** What are mostly studied research topics and the common challenges relevant to quality assurance of deep learning?
- **RQ-2:** What is secure deep learning engineering and its future direction in perspective of quality assurance?

From the RQ-1, we intend to identify the mostly concerned topics in the research community and their common challenges, while RQ-2 concerns the key activities in SDLE life cycle, based on which we discuss our vision and future opportunities.

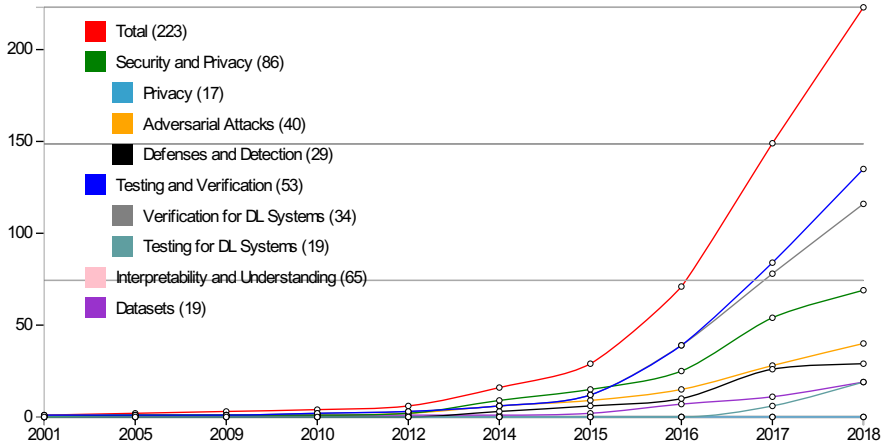

Fig. 1: The accumulative number of selected publications over years

2.2 Data Collection Methodology

Extensive research contributions are made on deep learning over the past decades, we adopt the following procedure to select works most relevant to the theme of this work.

- We first collect papers from conferences listed on the *Computer Science Rankings* within the scope of AI & machine learning, software engineering, and security.[3] To automate the paper collection procedure, we develop a Python-based crawler to extract paper information of each listed conference since the year 2000 and filter with keywords.
- To further reduce the search space for relevant topics, we use keywords (e.g., deep learning, AI, security, testing, verification, quality, robustness) to filter the collected papers.
- Even though, scraping all the listed conferences may still be insufficient, we therefore crawl outwards – extract all the related work for each keyword-filtered paper and crawl one level down of these papers.
- This finally results in 223 papers and we manually confirmed and labeled each paper to form a final categorized list of literature.

Paper Category and Labeling. To categorize the selected papers, we perform paper clustering by taking into account the title, abstract, and listed keywords. Based on further discussion of all authors (from both academia and industry with AI, SE, and security background), we eventually identify four main paper categories, and seven fine-grained categories in total (see Figure 1). In the next step, each paper is manually labelled into a target category for further analysis.

The Dataset and the Trend. Figure 1 shows the general trends of publication on secure deep learning research area, where the publication number (i.e., both total paper as well as in each category) dramatically increases over years. Such booming trend

[3] `http://csrankings.org/#/index?all`

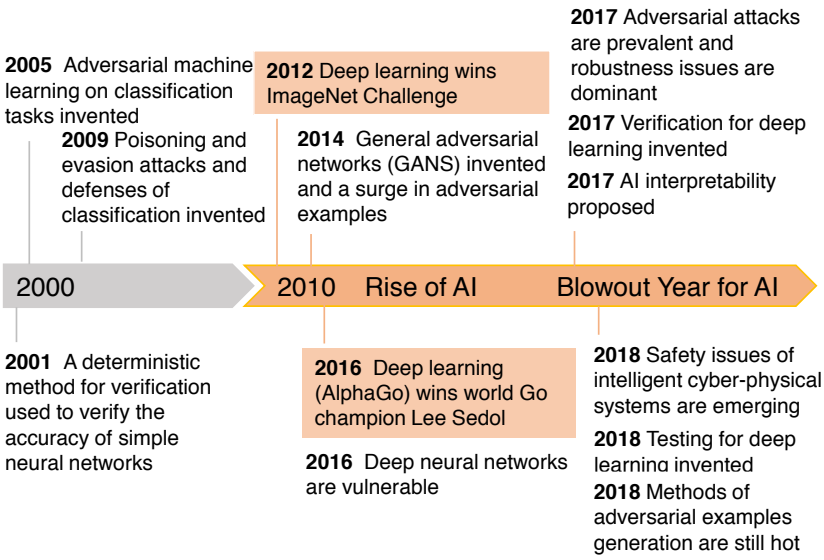

Fig. 2: Milestones of deep learning engineering relevant to security and software quality.

becomes even more obvious accompanied with the milestones of DLs (e.g., DL won ImageNet Challenge in 2012, AlphaGo defeated human championship in 2016), which is highlighted in Fig. 2. For the four main categories, we find the most publications are relevant to Security and Privacy (SP, 86 papers), followed by Interpretability and Understanding (IU, 65), Testing and Verification (TV, 53), and Datasets (17).

The SP category with the highest paper publication number is not surprising. Since Goodfellow et al. [20] posted the security issues of DLs, it attracted both the AI and security communities to escalate and burst a research competition on defending and attacking techniques. Even though, it still lacks a complete understanding of why current DL systems are still vulnerable against adversarial attacks. This draws the attention of researchers on interpreting and understanding how DL works, which would be important for both application and construction of robust DLs. As the recent emerging investment blowout in DL applications to safety-critical scenarios (e.g., autonomous driving, medical diagnose), its software quality has become a big concern, where researchers find that the different programming paradigm of DL makes existing testing and verification techniques unable to directly handle DLs [25, 30, 35]. Therefore, we have observed that many recent works are proposing novel testing and verification techniques for DLs, from testing criteria, test generation techniques, test data quality evaluation, to static analysis. Meanwhile, the dataset benchmarks of different DL application domains emerge to grow as well [15, 24, 37, 42], in order to facilitate the study of solving domain-specific problems by DLs (e.g., image classification, 3D object recognition, autonomous driving, skin disease classification).

Common Issues. In contrast to traditional software of which the decision logic is mostly programmed by human developers, deep learning adopts a data-driven programming paradigm. Specifically, a DL developer's major effort is to prepare the train-

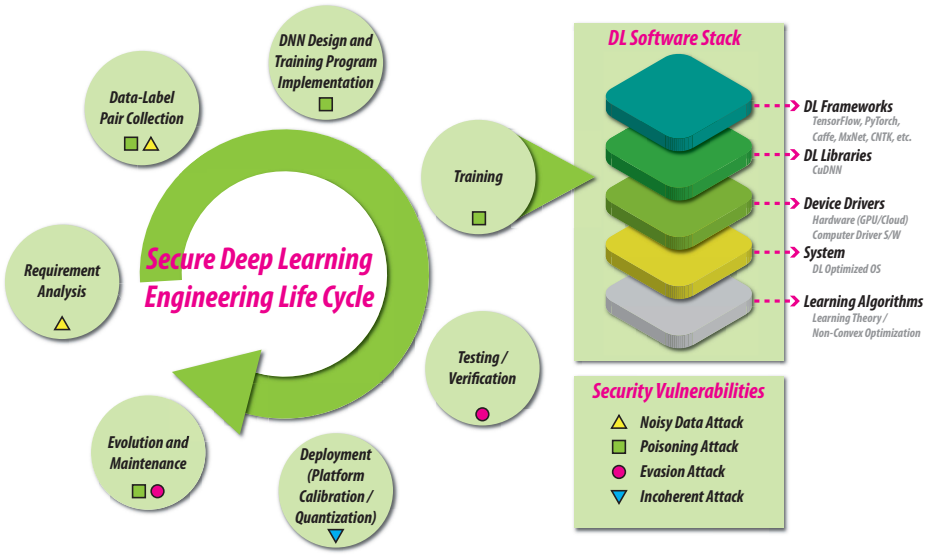

Fig. 3: Secure deep learning engineering life cycle

ing data (including knowledge to resolve a task) and neural network architecture, after which the decision logic is automatically obtained through the training procedure. On one hand, this paradigm reduces the burden of a developer who manually crafts the decision logic. On the other hand, for a DL developer, the logic training procedure is almost like a magic-box driven by an optimization technique. Due to the decision logic of DL is encoded into a DNN with high dimensional matrices, the interpretation and understanding, training procedure, as well as the obtained decision logic are all very difficult [28], which could be a root cause and a common challenge among all categories. For example, without completely understanding the decision logic of DL, it is hard to know in what particular case an adversarial attack could penetrate, and how we could defend against such attacks. In the case of testing, extensive studies are performed on analysis of traditional software bugs, their relations to software development activities, as well as techniques for defect detection. However, a comprehensive empirical study and understanding on why DL bugs occur still could not be well explained, let alone the root case analysis.

3 Secure Deep Learning Engineering Life Cycle

Due to the fundamental different programming paradigms of deep learning and traditional software, the secure deep learning engineering practice and techniques are largely different with traditional software engineering, although the major life cycle phases could still be shared.

We define *Secure Deep Learning Engineering (SDLE) as an engineering discipline of deep learning software production, through a systematic application of knowledge, methodology, practice on deep learning, software engineering and security, to requirement analysis, design, implementation, testing, deployment, and maintenance of deep learning software.*

Figure 3 shows the key life cycle phases of SDLE. In the rest of this section, we first describe each of the key development phases, their uniqueness and difference compared with traditional practices in software engineering, and then we discuss the security issues in current SDLE. In the next section, we explain the quality assurance necessity in SDLE life cycle, and highlight the challenges and opportunities.

Requirement Analysis. Requirement analysis investigates the needs, determines, and creates detailed functional documents for the DL products. DL-based software decision logic is learned from the training data and generalized to the testing data. Therefore, the requirement is usually measured in terms of an expected prediction performance, which is often a statistics-based requirement with uncertainty, as opposed to the rule-based one in traditional SE.

Data-Label Pair Collection. After the requirements of the DL software become available, a DL developer (potentially with domain experts for supervision and labeling) tries to collect representative data that incorporate the knowledge on a specific target task. For traditional software, a human developer needs to understand the specific task, figures out a set of algorithmic operations to solve the task, and programs such operations in the form of source code for execution. On the other hand, one of the most important sources of DL software is training data. The DL software automatically distills the computational solutions of a specific task under a designed neural network architecture.

DNN Design and Training Program Implementation. When the training data become available, a DL developer designs the DNN architecture, taking into account of requirement, data complexity, as well as the problem domain. For example, when addressing a general-purpose image processing task, convolutional layer components are often included in the DNN model design, while recurrent layers are often used to process tasks that has sequential inputs (e.g., natural language processing, speech recognition). To concretely implement the desired DNN architecture, a DL developer often leverages an existing DL framework to encode the designed DNN into a training program. Furthermore, the runtime training behaviors are also needed to be specified through the APIs provided by the DL framework (e.g., training epochs, learning rate, GPU/CPU configurations).

Runtime Training. After the DL programming ingredients (i.e., training data and training program) are ready. The runtime training procedure starts and systematically evolves the decision logic learning towards effectively resolving (e.g., classification, numerical prediction, synthesis&generation) a target task. The training procedure and training program adjustment might go back-and-forth several rounds until a satisfying performance is achieved. Although the training program itself is often written as traditional software (e.g., in `Python`, `Java`, `C++`), the obtained DL software is often encoded as a DNN model, consisting of the DNN architecture and weight matrices. The training process plays a central role in the DL software learning, to distill knowledge and solution

from the sources. It involves quite a lot of software and system engineering effort to realize the learning theory to DL software (see Figure 3) over years.

Testing & Verification. When the DNN model completes training with its decision logic determined, it goes through the systematic evaluation of its generality and quality through testing (or verification). Note that the testing activity in the AI community mainly considers whether the obtained DL model generalizes to the prepared test dataset, to obtain high test accuracy. On the other hand, the testing activity (or verification) in SDLE considers a more general evaluation scope, such as generality, robustness, defect detection, as well as other nonfunctional requirement (e.g., efficiency). The early weakness detection of the DL software provides valuable feedback to a DL developer for solution enhancement [47].

Deployment. A DL software passed the testing phase reaches a certain level of quality standard, and is ready to be deployed to a target platform. However, due to the platform diversity, DL framework supportability, and computation limitations of a target device, the DL software often needs to go through the platform calibration (e.g., compression, quantization, DL framework migration) procedure for deployment on a target platform. For example, once a DL software is trained and obtained on the `Tensorflow` framework, it needs to be successfully transformed to its counterpart of `TensorflowLite` (resp. `CoreML`) framework to `Android` (resp. `iOS`) platform. It still needs to go through on-device testing after deployment [22]. The testing during deployment not only considers the potentially incorrect behaviors that could be triggered at runtime, but also considers the behavior differences before and after deployment. In particular, whether a deployed version is an intended version for runtime execution. Due to the difference in platform, deep learning framework as well as hardware resources before and after deployment, systematically testing and providing feedback on the behavior changes of a deployed deep learning software would assist a DL developer to further enhance its quality.

Evolution and Maintenance. After a DL product is deployed, it might experience the procedure of modification for bug correction, performance and feature enhancements, or other attributes. The major effort in evolution and maintenance phases relies on the manual revision on design, source code, documentation, or other software artifacts. On the other hand, DL software focuses more on comprehensive data collection, DL model continuous learning (e.g., re-fitting, retro-fitting, fine-tuning, and re-engineering). For example, it is not uncommon that the new data are continuously collected that contain more domain-specific information of a particular task. A DL developer often considers how to incorporate the knowledge of such data into a target DL system to further enhance a DL software. During such a phase, a DL product would experience evolution, to update the feature, fix bugs, enhance robustness, etc. However, such a procedure are mostly driven by training data, which guides the direction of a DL produce enhancement. Furthermore, special engineering methods to better manage the version variants of a DL product (e.g., DL training program, models), as well as the data, would also be necessary, which to a large extent differs to those in the traditional software evolution and maintenance.

Security Issues in DL. The current practice of security in deep learning has fallen into the trap that many other domains have experienced. Almost every month new at-

tacks are identified [9, 12, 14, 20, 32, 33, 46] followed by new countermeasures [34, 45] which are subsequently broken [12, 23], and so on ad-infinitum. There is a broad and pressing need for a frontier-level effort on trustworthiness and security in DL to break this cycle of attacks and defenses. We have a unique opportunity at this time—before deep learning is widely deployed in critical systems—to develop the theory and practice needed for robust learning algorithms that provide rigorous and meaningful guarantees. If we rethink the SDLE life cycle (see Figure 3), security vulnerabilities can happen in almost every step. For instance, for the training related steps such as *Requirement Analysis, Data-Label Pair Collection* and *DNN design and training*, poisoning attacks can easily happen via manipulating training data. In the testing related steps, such as *testing & verification*and *deployment*, evasion attacks can take place by perturbing the testing data slightly (e.g. adversarial examples). In addition, when deploying the DL software to different platforms or with different implementation frameworks, there will always be opportunities for adversaries to generate attacks from one to the other.

We believe many of these security issues are highly intertwined the quality of current DL software, lacking systematic quality assurance solutions over the entire SDLE process which is largely missed in research works as described in the next section.

4 Towards Future Quality Assurance of SDLE

Over the past decades, software quality assurance discipline [36, 38] has been well-established for traditional software, with many experiences and practices widely applied in the software industry. However, the fundamentally different programming paradigm and decision logic representation of DL software make existing quality assurance techniques unable to be directly applied, forcing us to renovate the entire quality assurance procedure for SDLE. In this section, we pose our vision and challenges on quality assurance in SDLE to guide future research.

From the very beginning of SDLE, we need to rethink how to accurately define, specify, and document the of DL software requirement, especially for the functional requirements. This leaves us a question of whether we should follow a statistical-based approach, a rule-based approach, or their combination, which has not been well investigated yet.

The data play a key role in shaping the learning process and DL decision logic. However, most current research treats the data (e.g., training data) as high quality for granted, without a systematic quality control, inspection, and evaluation process. As poisoning attacks show, many incorrect behaviors and security issues could be introduced with the maliciously tweaked training data. How to select a suitable size while representative data would be an important question. In addition, data supervision and labeling process are also labor-intensive and error-prone. For example, ImageNet dataset contains more than one million general-purpose images. We also need to provide assistance and quality control for the data management (e.g., labeling, versioning, denoising) procedure.

It becomes even more challenging, when it comes to the implementation of the training program and framework. Most state-of-the-art DL frameworks are implemented as traditional software on top of the DL software stack. Even the learning theory is perfect,

it still has a big gap to transfer such ideally designed DL models to a DL application encoded on top of the DL framework. One big challenge is how to ensure the software stack (e.g., hardware drivers, DL library, DL framework) correctly implements the learning algorithm.

Another challenge is to provide useful interactive support to debug and visualize the training process. Most current DL training procedure only shows training loss (accuracy), validation loss (accuracy), which is mostly a black box to a DL developer. When the training procedure goes beyond expectation, the root-cause analysis becomes extremely difficult, which may come from the DL architecture issue, training program implementation issue, or the hardware configuration issue. Hence, the software engineering community needs to consider providing the novel debugging, runtime monitoring, and profiling support for the training procedure, which is involved with non-determinism and runtime properties hard to specify.

The large input space has already been a challenge for testing and verifying traditional software. Such a challenge is further escalated for DL software, due to its high dimensional input space and the internal latent space. Even though, traditional software testing has already explored many testing criteria as the goal to guide testing. How to design suitable testing criteria to capture the testing confidence still remains unclear. Even with some preliminary progress on testing criteria designed for DLs [27, 29, 30, 35], there are many more testing issues needed to be addressed, such as how to effectively generate tests [17, 29, 43, 44], how to measure the test data quality [31], and how to test DL robustness and vulnerabilities [10, 11]. More in-depth empirical studies that uncover the unique issues (e.g., [48]) of SDLE are also necessary to provide insight and guidance to build deep learning systems with better quality and reliability.

Further DL challenge comes up with current deployment process: (1) target device computation limitations, and (2) DL framework compatibility across platforms and frameworks. The DL software is mostly developed and trained on the cloud or PCs with powerful GPU support. When it needs to be deployed on a mobile device or edge-computing device with limited computation power, the DL software must be optimized or quantized for computation/energy efficiency, which could introduce defects or behavior differences. How to ensure the quality and detect the potential issues during this process is an important problem. In addition, the current DL frameworks might not always be supported by different platforms. For example, the TensorFlow is not directly supported by Android or iOS, and how to make DL software cross-platform compatible would be an important direction.

Last but not least, the quality assurance concerns in DL software evolution and maintenance are mostly focused on avoiding introducing defects during change, which might rely on regression testing. However, how to effectively evolve the DL software and how to engineer the artifacts (e.g., data, training program, DL model) of a DL product during evolution still remains unknown, which we leave as an open question for further study.

5 Conclusion

Considering deep learning is likely to be one of the most transformative technologies in the 21st century, it appears essential that we begin to think about how to design fully-fledged deep learning systems under a well-tested development discipline. This paper defines the secure deep learning engineering and discusses the current challenges, opportunities, and puts forward open questions from the quality assurance perspective, accompanied by a paper repository. We hope our work can inspire future studies towards constructing robust, reliable and safe DL software with high quality.

Acknowledgments

We thank Felix Juefei-Xu, Xiaofei Xie, Minhui Xue, Qiang Hu, Xiaoning Du, Yi Li, Sen Chen, Bo Li, Jianxiong Yin, Simon See for their contribution to initiate the early work of this paper. We also acknowledge the support of NVIDIA AI Tech Center (NVAITC) to our research, which largely shapes the direction of this work. This research was supported (in part) by the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), National Satellite of Excellence in Trustworthy Software System (Award No. NRF2018NCR-NSOE003-0001) administered by the National Cybersecurity R&D Directorate; JSPS KAKENHI Grant NO.19H04086, NO.18H04097, and Qdai-jump Research Program NO.01277.

References

1. BBC: Google's DeepMind to peek at NHS eye scans for disease analysis (2016), https://www.bbc.com/news/technology-36713308
2. BBC: AI image recognition fooled by single pixel change (2018), https://www.bbc.com/news/technology-41845878
3. BBC: Artificial intelligence 'did not miss a single urgent case' (2018), https://www.bbc.com/news/health-44924948
4. BBC: Can we trust AI if we don't know how it works? (2018), https://www.bbc.com/news/business-44466213
5. BBC: General Motors and Fiat Chrysler unveil self-driving deals (2018), https://www.bbc.com/news/business-44325629
6. BBC: Google cars self-drive to Walmart supermarket in trial (2018), https://www.bbc.com/news/technology-44957251
7. BBC: Honda to invest $2.8bn in GM's self-driving car unit (2018), https://www.bbc.com/news/business-45728169
8. BBC: Jaguar self-drive car revealed in New York (2018), https://www.bbc.com/news/technology-43557798
9. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 387–402. Springer (2013)
10. Breier, J., Hou, X., Jap, D., Ma, L., Bhasin, S., Liu, Y.: DeepLaser: Practical Fault Attack on Deep Neural Networks. ArXiv e-prints

11. Breier, J., Hou, X., Jap, D., Ma, L., Bhasin, S., Liu, Y.: Practical fault attack on deep neural networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS '18 (2018)
12. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: Security and Privacy (SP), IEEE Symposium on. pp. 39–57 (2017)
13. Chen, C., Seff, A., Kornhauser, A., Xiao, J.: Deepdriving: Learning affordance for direct perception in autonomous driving. In: 2015 IEEE International Conference on Computer Vision (ICCV). pp. 2722–2730 (Dec 2015). https://doi.org/10.1109/ICCV.2015.312
14. Chen, P.Y., Sharma, Y., Zhang, H., Yi, J., Hsieh, C.J.: Ead: Elastic-net attacks to deep neural networks via adversarial examples. arXiv preprint arXiv:1709.04114 (2017)
15. Chen, Y., Wang, J., Li, J., Lu, C., Luo, Z., Xue, H., Wang, C.: Lidar-video driving dataset: Learning driving policies effectively. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
16. Doshi-Velez, F., Kim, B.: Towards A Rigorous Science of Interpretable Machine Learning. ArXiv e-prints
17. Du, X., Xie, X., Li, Y., Ma, L., Liu, Y., Zhao, J.: 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. pp. 477–487. ESEC/FSE 2019 (2019)
18. Elgammal, A.M., Liu, B., Elhoseiny, M., Mazzone, M.: CAN: creative adversarial networks, generating "art" by learning about styles and deviating from style norms. CoRR **abs/1706.07068** (2017), http://arxiv.org/abs/1706.07068
19. Eliot, L.B.: Advances in AI and Autonomous Vehicles: Cybernetic Self-Driving Cars Practical Advances in Artificial Intelligence (AI) and Machine Learning. LBE Press Publishing, 1st edn. (2017)
20. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. ICLR (2015)
21. Google Accident: A Google self-driving car caused a crash for the first time (2016), https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report
22. Guo, Q., Chen, S., Xie, X., Ma, L., Hu, Q., Liu, H., Liu, Y., Zhao, J., Li, X.: An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In: Proceedings of the 34rd ACM/IEEE International Conference on Automated Software Engineering. ASE 2019
23. He, W., Wei, J., Chen, X., Carlini, N., Song, D.: Adversarial example defenses: Ensembles of weak defenses are not strong. arXiv preprint arXiv:1706.04701 (2017)
24. Huang, X., Wang, P., Cheng, X., Zhou, D., Geng, Q., Yang, R.: The ApolloScape Open Dataset for Autonomous Driving and its Application. ArXiv e-prints
25. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: International Conference on Computer Aided Verification. pp. 3–29 (2017)
26. Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., Sayres, R.: Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). ArXiv e-prints
27. Kim, J., Feldt, R., Yoo, S.: Guiding deep learning system testing using surprise adequacy **arXiv:1808.08444** (2018)
28. Lipton, Z.C.: The mythos of model interpretability. CoRR **abs/1606.03490** (2016), http://arxiv.org/abs/1606.03490
29. Ma, L., Juefei-Xu, F., Xue, M., Li, B., Li, L., Liu, Y., Zhao, J.: Deepct: Tomographic combinatorial testing for deep learning systems. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 614–618 (Feb 2019)

30. Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J., Wang, Y.: Deepgauge: Multi-granularity testing criteria for deep learning systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 120–131. ASE 2018 (2018)
31. Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., et al.: Deepmutation: Mutation testing of deep learning systems. The 29th IEEE International Symposium on Software Reliability Engineering (ISSRE) (2018)
32. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. arXiv preprint arXiv:1511.04599 (2015)
33. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: Security and Privacy (EuroS&P), 2016 IEEE European Symposium on. pp. 372–387. IEEE (2016)
34. Papernot, N., McDaniel, P.D., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: IEEE Symposium on Security and Privacy, SP 2016. pp. 582–597 (2016)
35. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 1–18 (2017)
36. Pressman, R.: Software Engineering: A Practitioner's Approach. McGraw-Hill, Inc., New York, NY, USA, 7 edn. (2010)
37. Ramanishka, V., Chen, Y.T., Misu, T., Saenko, K.: Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
38. Ruparelia, N.B.: Software development lifecycle models. SIGSOFT Softw. Eng. Notes **35**(3), 8–13 (May 2010). https://doi.org/10.1145/1764810.1764814
39. The New York Times: Alexa and Siri Can Hear This Hidden Command. You Can't (2018), https://www.nytimes.com/2018/05/10/technology/alexa-siri-hidden-command-audio-attacks.html
40. The New York Times: Toyota, SoftBank Setting Up Mobility Services Joint Venture (2018), https://www.nytimes.com/aponline/2018/10/04/world/asia/ap-as-japan-toyota-softbank.html
41. Uber Accident: After Fatal Uber Crash, a Self-Driving Start-Up Moves Forward (2018), https://www.nytimes.com/2018/05/07/technology/uber-crash-autonomous-driveai.html
42. Xiang, Y., Kim, W., Chen, W., Ji, J., Choy, C., Su, H., Mottaghi, R., Guibas, L., Savarese, S.: Objectnet3d: A large scale database for 3d object recognition. In: European Conference Computer Vision (ECCV) (2016)
43. Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S.: Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 146–157. ISSTA 2019 (2019)
44. Xie, X., Ma, L., Wang, H., Li, Y., Liu, Y., Li, X.: Diffchaser: Detecting disagreements for deep neural networks. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence (2019)
45. Xu, W., Evans, D., Qi, Y.: Feature squeezing: Detecting adversarial examples in deep neural networks. arXiv preprint arXiv:1704.01155 (2017)
46. Xu, W., Qi, Y., Evans, D.: Automatically evading classifiers. In: Proceedings of the 2016 Network and Distributed Systems Symposium (2016)
47. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine Learning Testing: Survey, Landscapes and Horizons. arXiv e-prints (Jun 2019)

48. Zhang, T., Gao, C., Ma, L., Lyu, M.R., Kim, M.: An empirical study of common challenges in developing deep learning applications. The 30th IEEE International Symposium on Software Reliability Engineering (ISSRE) (2019)