WASP Software Engineering Assignment

Yiran Wang
Linköping University, Sweden
yiran.wang@liu.se

My Ph.D. project is centered around the intersection between software engineering (SE) and machine learning (ML). The primary objective is to enhance the productivity and quality of ML programs by detecting defects early in the development phase, before the code is executed.

Recently, ML has become a popular technical approach and has shown promising outcomes in various domains, including self-driving cars, large language models, image generation and recognition, and healthcare. Nevertheless, ML development presents challenges. ML programs are usually written in dynamic languages such as Python, and constructed by third-party ML libraries. The misinterpretation and misuse of ML libraries increase the risk of defects in the code. Moreover, many ML practitioners in different fields may lack a background in software development, further contributing to the prevalence of defects.

Furthermore, training and testing ML models often require significant computational resources. Defects in ML programs can lead to failures during prolonged execution, resulting in wasted resources and reduced productivity. ML bugs can also go silent and remain undetected during dynamic testing, which can result in unexpected results and pose threats to safety-critical systems. To address these concerns, an effective and efficient approach is needed for static detection of ML defects during the early development stage, while coding before execution. Static analysis presents opportunities in this regard, although current static analysis tools for ML programs have certain limitations. For example, they struggle with static type inference for dynamic languages and operators in third-party or ML libraries, and handling incomplete data parameters (e.g. dimensions, shapes) that are supported by operators in ML libraries. Thus, it is of great interest to research static analysis for ML programs.

Topic 1: SE for ML

In recent years, there has been active discussion on adjusting SE best practices for ML systems to reduce technical debt and improve the quality of ML programs. Recent studies have highlighted the specific needs of ML program requirements in the context of SE system design [1].

In relation to my research, testing has always been an essential step in SE processes and application workflows. However, unlike traditional software, ML systems often exhibit complex inter-dependencies between components, possess a statistical nature, lack clear testing rules, involve problem-specific architectures, and are affected by the unavoidable inconsistency between real-world data and training data. These aspects make dynamic testing less reliable for quality and correctness assurance. On the other hand, static analysis provides an opportunity for thorough testing of ML systems without code execution, thus avoiding the wastage of computational resources.

Topic 2: Verification and Validation

Verification involves confirming whether a specific function adheres to a set of well-designed rules. However, for ML systems, the statistical nature and heavy dependency to the data of ML models make the verification tests more challenging, as ML models often produce results with varying probabilities and the results can be different even for the same input. Validation, on the other hand, focuses on assessing whether the results fulfill the requirements of the designers or users to a certain extent, which could be potentially used in ML system testing.

However, traditional verification and validation tests may overlook silent defects that manifest under certain inputs or reveal themselves when the distribution of the input data shifts, resulting in unintentional results that are difficult to troubleshoot. By incorporating static analysis in the development phase, such defects can potentially be defected and fixed, reducing the technical debts of ML systems.

Topic 3: Productivity

The objective of my research project is to create an open-source tool that integrates with ML development platforms, such as notebook applications (e.g. Jupyter Notebook, JupyterLab) and script editors. The tool aim to provide real-time defect detection assistance to ML developers during coding, without requiring code execution. The tool will be designed to be low-cost to install and use, scalable, and non-disruptive to the usual workflow, effectively promoting the productivity of ML development.

Topic 4: Explainability

My research project will combine abstract interpretation-based and ML-based static analysis methods for the early detection of defects in ML programs. Abstract interpretation-based techniques can analyze and detect faults comprehensively but may generate false positives. On the other hand, ML-based techniques might produce fewer false alarms and require less analysis time, but they could overlook critical problems.

Moreover, ML-based techniques often lack explainability. However, as static analyzers, presenting the detected faults with the reasons is crucial for efficient troubleshooting and fixes by developers. Thus, incorporating abstract interpretation-based and ML-based static analysis methods will enhance the effectiveness of static analyzers without compromising the important aspect of explainability.

Topic 5: Quality Assurance

Quality assurance is a crucial process in the software development workflow, ensuring that the final output aligns with the required standards and achieves the desired quality levels. As the use of ML systems grow at a rapid rate, as well as expanding into new industries, the importance of a robust quality assurance becomes evermore pressing. At the same time, the large variety in ML systems, which will only increase as many industries implements ML to fit with their existing architecture, posing significant challenges in establishing a unified framework for quality assurance.

Through the research that I will carry out during my Ph.D. studies, the development of advanced methods and cutting-edge tools for static analysis of ML programs will have a profound impact on the quality assurance process. As previously mentioned, current static analysis tools encounter difficulties in detecting many common bugs and errors in ML programs. Consequently, the burden on the quality assurance process to identify and address these defects is substantially amplified.

Two of the critical aspects during quality assurance for industrial systems, as highlighted by Fujii et al. [2], is the evaluation of model robustness and system quality. However, without a proper static analysis process, certain errors, especially those related to parameters and dimensions, may go unnoticed during development. This can lead to a functioning program that runs without crashing but fails to produce the correct outputs, which can then be misinterpreted by individuals or systems that rely on this information for making decisions. Consequently, the evaluation of model robustness and system quality during quality assurance demands extensive testing to ensure the absence of such elusive errors, some of which can be exceedingly challenging to detect within complex ML programs.

By providing an effective tool for static analysis, my research can significantly reduce, if not altogether eliminate, the likelihood of many types of errors going unnoticed during development. As a result, the quality assurance process would no longer require as many resources to detect and troubleshoot these defects, allowing for the allocation of more time and resources to other crucial aspects of quality assurance. The early detection and resolution of defects through static analysis would ultimately lead to enhanced product quality and reliability, minimizing the risk of errors causing significant disruptions or consequences in real-world applications.

Topic 6: Automated Software Testing

Automated software testing is a cornerstone of modern software development, offering an automated approach to quality assurance. It seeks to streamline and release human effort by integrating standardized automated tools into development workflow, validating and verifying essential aspects of a software product and make sure it meets predefined requirements. It usually includes designing test scripts and repetitively executing test cases. Automated software testing is particularly advantageous for large-scale projects that involve a substantial number of clear-defined test cases. In the software development process, automated software testing is often integrated into the continuous delivery pipeline. In the meanwhile, static analysis contributes significantly to the automated software testing process.

In the context of ML systems, testing and validating these complex models present unique challenges. Unlike traditional software, ML systems are not solely determined by the software code but are heavily influenced by the data on which they are trained [3]. This data-dependent nature introduces complexities in crafting effective automated tests, as it requires generating realistic input data or creating test environments that accurately mimic real-world scenarios.

Moreover, as stated earlier, ML programs have different standards and quality requirements compared to traditional software, and this sometimes necessitates frequent manual check during automated software testing process. Consequently, substantial human effort and additional computational resources are required to run necessary customized downstream tests.

Riccio et al. [3] presents the challenges of integrating automated software testing into the automated testing process of ML development. Some of the key challenges include automated generation of realistic input data or test environments, creating unified test oracles, and defining reliable evaluation metrics. These challenges further underscore the need for a proactive approach to defect detection in ML programs, which can be effectively addressed through static analysis during the early stages of development.

In my research, by incorporating static analysis into the early development phase, specifically during the coding process before code execution, ML defects can be more reliably and efficiently detected. This proactive approach is especially valuable as the coding environment and associated information are readily available, offering developers a comprehensive view of potential issues and providing detailed information in fixing the problems at an early stage.

Moreover, the application of ML methods in automated traditional software testing also presents an exciting avenue for enhancing testing efficiency. Additionally, ML-based approaches may offer novel solutions to reduce human workload during the preparation phase of automated testing, such as test case design and development [4], and test software applications (e.g. games) with a sequential decision process [5]. Existing studies have demonstrated promising results in utilizing ML algorithms for generating, refining, and evaluating test cases, as well as predicting the cost of test activities [4]. However, there are concerns regarding the explainability of ML models and the quality of the training data, as data plays a pivotal role in the effectiveness of ML models.

My research includes exploring how ML methods can be incorporated with traditional static analysis to address these concerns effectively. By leveraging the power of ML techniques, we can reduce analyzing time and alleviate the need for manual rule extraction in static analysis. By combining static analysis and ML-based approaches, we can create a comprehensive and efficient testing framework that leverages the strengths of both methodologies. This integrated approach has the potential to significantly enhance the testing process, leading to improved software reliability, reduced defect rates, and enhanced overall software quality. Furthermore, as static analysis is performed early in the development process, it serves as a vital component of a comprehensive quality assurance strategy, contributing to the successful deployment of robust and dependable ML systems in real-world applications.

Future trends of SE for ML

Over the next five to ten years, ML/AI Engineering is expected to experience significant growth and gain great importance across diverse industries and domains. We can anticipate the establishment of a more cohesive and dependable standard and framework for deploying ML systems in production, further enhancing its integration into various sectors with a strong emphasis on responsible AI practices. The convergence of ML/AI with other cutting-edge technologies will unlock fresh opportunities and lead to breakthroughs in multiple fields. Meantime, these experiences and more newly generated data will drive advancements in ML/AI research itself, pushing the boundaries of its capabilities. Overall, ML/AI Engineering will assume a pivotal role in shaping a responsible and innovative future for the field of ML/AI.

References

[1] S. Amershi, A. Begel, C. Bird, *et al.*, "Software Engineering for Machine Learning: A Case Study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2019, ISBN: 978-1-72811-760-7. DOI: 10.1109/ICSE-SEIP.2019.00042.

[2] G. Fujii, K. Hamada, F. Ishikawa, *et al.*, "Guidelines for Quality Assurance of Machine Learning-Based Artificial Intelligence," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, 2020, ISSN: 0218-1940, 1793-6403. DOI: 10.1142/S0218194020400227.

[3] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: A systematic mapping," *Empirical Software Engineering*, vol. 25, 2020, ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-020-09881-0.

[4] V. H. S. Durelli, R. S. Durelli, S. S. Borges, *et al.*, "Machine Learning Applied to Software Testing: A Systematic Mapping Study," *IEEE Transactions on Reliability*, vol. 68, 2019, ISSN: 0018-9529, 1558-1721. DOI: 10.1109/TR.2019.2892517.

[5] Y. Zheng, X. Xie, T. Su, *et al.*, "Wuji: Automatic Online Combat Game Testing Using Evolutionary Deep Reinforcement Learning," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, ISBN: 978-1-72812-508-4. DOI: 10.1109/ASE.2019.00077.