Session 6: Maintenance of AI-based Systems ICSME 2024

André Butuc
up202400040

José Mendes
up202402636

Ubaldo Senete
up202402696

I. Introduction

The 40th IEEE International Conference on Software Maintenance and Evolution (ICSME 2024) occurred from October 6 to 11, 2024, in Flagstaff, Arizona, USA. This premier event brought together researchers and practitioners from academia, industry and government to discuss the latest innovations, trends, experiences, and challenges in software maintenance and evolution.

The conference featured a diverse program, including keynote speeches, research presentations, industry talks, tool demonstrations, and workshops. Keynote speakers addressed evolving software engineering practices for AI-enabled systems and the influence of AI-driven innovations on developer communities.

The sessions covered a wide range of topics, including code understanding and optimization, software architecture and design, developer experience and communication, and software development practices and tools. The event also emphasized emerging areas like the maintenance and evolution of AI-based applications and the use of large language models in software evolution tasks.

This paper will provide a summarized overview of the ”Session 6: Maintenance of AI-based Systems at Fremont” session, chaired by Sujoy Roychowdhury (Ericson R&D), that includes a total of four research track papers, one NIER paper and one industry track paper.

II. Main Issues

The main focus of the session is to expose and address problems as well as propose solutions regarding the Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) realms. The main concerns are the following.

- How to test and debug ML projects?
- How to deal with ML technical debt?
- How to leverage ML models in terms of code maintainability and refactoring?
- How to reduce memory consumption on the execution of ML models?

III. Main Results

A. Testing and Debugging ML projects

Testing and debugging machine learning (ML) projects can be tough because of the complexity of the models and the large datasets.

To better understand these challenges, an empirical study was presented by [1] on property-based testing (PBT) within ML projects to identify issues related to this testing approach. Their work analyzed 58 GitHub repositories that depend on Hypothesis, a property-based testing library for Python, with a manual inspection of 28 projects. They found that developers hold mixed sentiments toward PBT: while it is praised for uncovering edge cases, challenges such as flaky tests and long execution times are common complaints. The study revealed that projects often rely on custom data generators (e.g., for tensors or ML-specific data structures), indicating a lack of domain-specific tooling. Notably, PBT is rarely applied to trained ML models (due to the oracle problem) and is instead used for testing preprocessing logic and training code. This work highlights gaps in PBT adoption for ML and calls for better tooling to support ML workflows.

To address the challenge of debugging, in the case of Deep Reinforcement Learning (DRL), [2] introduced RLExplorer, which tracks and analyzes traces (e.g., rewards, actions) to detect issues like action stagnation or abnormal entropy which are common bug threats. Tested on 11 real-world faulty DRL code samples from Stack Overflow, RLExplorer successfully diagnosed 83% of faults. In a human study with 15 DRL developers, it was shown to detect and fix bugs 3.6 times more effectively than manual debugging.

B. Technical Debt in ML projects

Deep Learning is becoming widely used in several areas where software quality is crucial; however, there is also an additional pressure to deploy these models which ultimately leads to less pondered decisions. But what are exactly the smells or malpractices developers are introducing that are causing ML code to accumulate technical debt?

[3], presented a work that defines a taxonomy of self-admitted technical debt (SATD) in Deep Learning (DL) Systems. These systems often accumulate technical debt due to rapid deployment pressures. The authors analyzed 443 SATD instances from 54 open-source DL projects, they identified 41 SATD types organized into 7 categories across two high-level dimensions: Infrastructure (e.g., hardware and API usage for developing the DL framework/application) and DL Life-Cycle (e.g., data handling, model issues, training process, inference process and pipeline management).

A key insight is that SATD in DL systems often reflects intentional trade-offs (e.g., using suboptimal hyperparameters for faster iteration) rather than accidental bugs. The authors also found that traditional static analysis tools (e.g., PyLint, Flake8) detected only 5% of SATD instances, highlighting the need for DL-specific detection mechanisms. This taxonomy provides a structured way to understand and prioritize technical debt in DL projects, offering actionable guidance for developers and researchers.

C. ML Efficiency on Code Maintainability and Refactoring

Besides the maintenance of AI-based systems, the session also focuses on how we can leverage AI tools to ensure code maintainability and facilitate bug fixing and refactoring.

[4] focuses on tackling the challenge associated with Python's type error bugs, specifically, the use of LLMs to generate patches.

LLMs require the input of buggy code snippets, as well as its surrounding context to generate patches; however, due to the finite input context window of these models, sometimes the length of the source code files surpasses the context window's capacity.

To tackle the challenge regarding context dimension, [4] proposed the RetypeR solution that uses two information retrieval stages: external and internal. The external retrieval stage aims at acquiring fix patterns from external code repositories containing historical bug fixes. The internal retrieval stage aims at extracting donor code (e.g. variable names) from local context within source files.

RetypeR successfully fixes 62 and 29 type errors from TypeBugs and BugsInPy, respectively, outperforming existing state-of-the-art approaches, such as TypeFix, ChatGPT and Codex.

Shifting to the application of ML models to improve code maintainability, [5] analyzed and compared different tools to set a benchmark for maintainability metrics, including CodeScene's Code Health, SonarQube, and Microsoft's Maintainability Index. The researchers found that CodeScene's Code Health metric performed just as well as top machine learning models and even beat the average human expert in predicting code maintainability. Unlike SonarQube, which often gave false positives, CodeScene provided more useful insights by identifying specific code issues and offering suggestions for improvement.

D. Reduce ML model memory consumption

DL models consume a lot of memory, mainly associated with the amount of parameters as well as their complex computation graphs. To cope with this memory footprint challenge, the academia and industry have introduced static memory computing methods, but also DL infrastructure, such as Apache TVM and Tensorflow, that consist of computation graphs compilers that make the graph less complex.

To address the memory consumption problem, [6], proposed the "OPASS" solution to optimize the TVM compiler. OPASS computes the upper and lower bounds of the computation graph's memory footprint and then optimizes the graph heuristically and iteratively. In each iteration, it picks up a pass by the explicit effect of memory footprint reduction and the implicit effects for further optimizations.

OPASS was evaluated on REBENCH, a self-developed benchmark by Pengbo Nie et al., and two real-world models, having achieved on average 51,16% of memory reduction, outperforming TVM's default sequence by 1.77x and also leading to extra memory reductions of 5-12%.

The results achieved by OPASS provide an important benchmark to enable DL model execution on memory-constrained devices.

IV. Closing Thoughts

The session successfully addressed the four main issues in maintaining AI-based systems, offering both practical tools and foundational insights.

Papers like RLExplorer and OPASS stood out for their direct applicability, providing concrete solutions for debugging reinforcement learning programs and reducing memory consumption in DL models. These tools highlight the value of domain-specific approaches to tackle AI's unique challenges. Equally important are contributions like the SATD taxonomy and maintainability benchmarks, which lay the groundwork for future research. While less immediately actionable, they help developers and researchers recognize pitfalls in AI system design and prioritize technical debt.

Together, these works reflect the dual need in AI maintenance: solving today's problems while preparing for tomorrow's challenges as AI systems grow in scale and complexity.

References

[1] C. Wauters and C. De Roover, "Property-based testing within ml projects: An empirical study," in *Proceedings of the 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2024, pp. 648–653. DOI: 10.1109/ICSME58944.2024.00067.

[2] R. Bouchoucha, A. H. Yahmed, D. Patil, *et al.* "Toward debugging deep reinforcement learning programs with rl-explorer." arXiv preprint arXiv:2410.04322. arXiv: 2410.04322 [cs.SE]. (2024), [Online]. Available: https://arxiv.org/abs/2410.04322.

[3] F. Pepe, F. Zampetti, A. Mastropaolo, G. Bavota, and M. D. Penta. "A taxonomy of self-admitted technical debt in deep learning systems." arXiv preprint arXiv:2409.11826. arXiv: 2409 . 11826 [cs.SE]. (2024), [Online]. Available: https://arxiv.org/abs/2409.11826.

[4] S. Hao, X. Shi, and H. Liu, "Retyper: Integrated retrieval-based automatic program repair for python type errors," in *Proceedings of the 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2024, pp. 199–210. DOI: 10.1109/ICSME58944.2024.00028.

[5] M. Borg, M. Ezzouhri, and A. Tornhill. "Ghost echoes revealed: Benchmarking maintainability metrics and machine learning predictions against human assessments." arXiv preprint arXiv:2408.10754. arXiv: 2408.10754 [cs.SE]. (2024), [Online]. Available: https://arxiv.org/abs/2408.10754.

[6] P. Nie, Z. Wang, C. Wan, *et al.*, "Opass: Orchestrating tvm's passes for lowering memory footprints of computation graphs," in *Proceedings of the 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2024, pp. 175–186. DOI: 10.1109/ICSME58944.2024.00026.