Software Engineering for AI Systems: Challenges of Developing, Testing, and Maintaining Machine Learning Systems Compared to Traditional Software

UDOKPORO JAMACHI BERNARD
De Montfort University

Abstract- ***The emergence of Artificial Intelligence (AI) and Machine Learning (ML), dates back between 1940's and 1950's. In recent years, there has been a surge in the quest/demand for applications that implement AI and ML technology. As with traditional development, software testing is a critical component of an efficient AI/ML application. However, the approach to development methodology used in AI/ML varies significantly from traditional development. Owing to these variations, numerous software developing and testing challenges occur. This paper aims to recognize and to explain some of the biggest challenges that software developers and testers face in maintaining and dealing with Artificial Intelligence (AI) / Machine Learning (ML) systems/applications compared to traditional software. The challenges in developing, testing, and maintaining machine learning (ML) systems compared to traditional software engineering, are due to the data-driven and adaptive nature of ML. For future research, this study has key implications. Each of the challenges outlined in this paper is ideal for further investigation and has great potential to shed light on the way to more productive software testing strategies and methodologies that can be applied to AI/ML applications.***

I. INTRODUCTION

Software systems with intelligent components based on machine learning (ML) techniques have been widely developed and are now applied in various fields, such as electronic commerce, finance, manufacturing, healthcare, entertainment, and the automotive industry. These practical applications (ML applications) have been anchored by significant advances in ML techniques and software platforms for ML development. ML techniques have been copiously researched and published over a broad range of topics. In particular, the breakthrough in deep learning research is the driving force behind the advance of ML techniques. Many papers on deep learning techniques, including learning algorithms, performance improvement, evaluations, and applications, have been extensively published.

However, the systematic development, deployment and operation of ML applications faces major difficulties. The methodologies and tools of software engineering (SE) have greatly contributed to a wide range of activities in the lifecycles of traditional information systems, but are difficult to implement in ML application projects because ML applications and traditional software systems differ in fundamental ways. An ML application involves at least a computational model (an ML model) which is trained on some training data, and which processes additional data to make some inferences. The behavior of an ML model-based program depends on the training data, and is often unpredictable. This phenomenon introduces various uncertainties into the system's outcomes. The lifecycle process of ML applications also differs from that of traditional software processes. Machine learning algorithms, models and related techniques are rapidly evolving and new challenges are emerging. Such situations make software engineering practices for ML applications more difficult activities.

II. DEVELOPMENT CHALLENGES

Data Dependency and Quality:
ML models heavily rely on large volumes of high-quality, relevant data for training. Data collection, cleaning, preprocessing, and feature engineering are time-consuming and critical steps. Poor data quality or insufficient data can significantly hinder model performance.

Iterative and Experimental Process:
ML development is often more iterative and experimental than traditional software, involving repeated cycles of model training, evaluation, and hyperparameter tuning. This makes project timelines less predictable.

Model Selection and Training:
Choosing the right ML model and optimizing its parameters involves experimentation and iteration,

contrasting with the deterministic algorithm design in conventional software.

Model Selection and Complexity:
Choosing the right ML algorithm and architecture for a given problem can be complex, requiring expertise in various techniques and understanding their trade-offs.

Explainability and Interpretability:
Many advanced ML models are "black boxes," making it difficult to understand their decision-making process, which is often a requirement in regulated industries or for debugging.

Version Control for Data and Models:
Managing versions of both code and the datasets and models that evolve during training adds complexity to version control systems.

Ethical Considerations and Bias:
ML models can inherit biases present in the training data, leading to unfair or discriminatory outcomes. Addressing bias and ensuring fairness requires careful consideration during development.

Hyperparameter Tuning:
Finding the optimal hyperparameters for Machine Learning (ML) model, can be time-consuming and require significant expertise.

III. TESTING CHALLENGES

Lack of Clear Requirements:
Machine Learning (ML) systems, often have ambiguous or uncertain requirements, making it difficult to define test cases and evaluate performance.

Non-Deterministic Behavior:
ML models can exhibit varying outputs for similar inputs due to their probabilistic nature, making traditional unit and integration testing methods less effective.

Non-Determinism and Generalization:
Unlike traditional software with predictable outputs for given inputs, ML models exhibit probabilistic behavior and need to generalize to unseen data. Testing requires evaluating performance on diverse datasets and assessing generalization capabilities.
Defining "Correct" Output:
For many ML applications, there isn't a single "correct" output, making traditional unit testing difficult. Evaluation relies on metrics like accuracy, precision, recall, or F1-score, which require large test sets.

Data Distribution Shift:
The distribution of data in production can change over time (data drift), causing model performance degradation. Testing strategies must account for this potential shift.

Interpretability and Explainability:
Understanding why an ML model makes a particular prediction can be challenging, especially for complex models, making debugging and root cause analysis more difficult.

Data Drift and Concept Drift:
Model performance can degrade over time as the real-world data deviates from the training data, requiring continuous monitoring and re-training.

Edge Cases and Adversarial Attacks:
Identifying and testing for rare or unexpected scenarios (edge cases) and potential malicious inputs (adversarial attacks) is more complex in ML systems.

Lack of Oracles:
Determining the "correct" output for a given input can be challenging, especially in subjective tasks like image recognition or natural language processing.

Evaluation Metrics:
Choosing the right evaluation metrics for ML systems can be difficult, and different metrics can lead to different conclusions.

IV. MAINTENANCE CHALLENGES

Continuous Monitoring and Retraining:
ML models require ongoing monitoring of performance and data quality, necessitating regular retraining to maintain accuracy and adapt to changing environments.

Model Drift and Retraining:
As data distributions evolve or real-world conditions change, ML models can "drift" and require retraining with new data to maintain

performance. This necessitates robust MLOps pipelines for continuous monitoring and retraining.

Version Control for Data and Models:
Managing different versions of data, trained models, and code is more complex than in traditional software, where only code versions are primarily tracked.

Model Deployment and Scalability:
Deploying and scaling ML models in production environments can be complex, involving specialized infrastructure and monitoring tools.

Bias and Fairness:
Ensuring fairness and mitigating bias in ML models is an ongoing ethical and technical challenge that requires continuous evaluation and intervention.

Resource Management:
Training and deploying ML models, especially deep learning models, can be computationally intensive, requiring significant hardware resources and efficient resource management.

Monitoring and Alerting:
Continuous monitoring of model performance in production is crucial to detect performance degradation or anomalies, triggering alerts for necessary interventions.

Security and Privacy:
Protecting sensitive training data and deployed models from security breaches and ensuring data privacy compliance are critical and complex aspects of ML system maintenance.

Model Updates:
ML models require regular updates to maintain performance, which can be time-consuming and require significant resources.

Explainability and Transparency:
ML models can be difficult to interpret, making it challenging to understand and explain their decisions.

V. KEY STRATEGIES FOR OVERCOMING THESE CHALLENGES

Agile Development Methodologies:
Adopting agile development methodologies can help teams respond quickly to changing requirements and iterate on ML models.

Collaboration between Data Scientists and Engineers:
Close collaboration between scientists and engineers is essential for developing, testing, and maintaining ML systems.

Continuous Monitoring and Testing:
Continuous monitoring and testing of ML systems can help identify issues early and ensure optimal performance.

Explainability and Transparency Techniques:
Using techniques like feature attribution and model interpretability can help improve understanding and trust in ML models.

VI. MACHINE LEARNING SYSTEMS VERSUS TRADITIONAL SOFTWARE

Since ML is part of software engineering (SWE), and software has been successfully used in production for more than half a century, some might wonder why we don't just take tried-and-true best practices in software engineering and apply them to ML.

That's an excellent idea. In fact, ML production would be a much better place if ML experts were better software engineers. Many traditional SWE tools can be used to develop and deploy ML applications.

However, many challenges are unique to ML applications and require their own tools. In SWE, there's an underlying assumption that code and data are separated. In fact, in SWE, we want to keep things as modular and separate as possible (see the Wikipedia page on separation of concerns).

On the contrary, ML systems are part code, part data, and part artifacts created from the two. The trend in the last decade shows that applications developed with the most/best data win. Instead of focusing on improving ML algorithms, most companies will focus on improving their data. Because data can change quickly, ML applications need to be adaptive to the changing environment, which might require faster development and deployment cycles.
In traditional SWE, you only need to focus on testing and versioning your code. With ML, we have to test

and version our data too, and that's the hard part. How to version large datasets? How to know if a data sample is good or bad for your system? Not all data samples are equal -- some are more valuable to your model than others. For example, if your model has already trained on one million scans of normal lungs and only one thousand scans of cancerous lungs, a scan of a cancerous lung is much more valuable than a scan of a normal lung. Indiscriminately accepting all available data might hurt your model's performance and even make it susceptible to data poisoning attacks.

The size of ML models is another challenge. As of 2022, it's common for ML models to have hundreds of millions, if not billions, of parameters, which requires gigabytes of random-access memory (RAM) to load them into memory. A few years from now, a billion parameters might seem quaint -- like, "Can you believe the computer that sent men to the moon only had 32 MB of RAM?"
However, for now, getting these large models into production, especially on edge devices, is a massive engineering challenge. Then there is the question of how to get these models to run fast enough to be useful. An autocompletion model is useless if the time it takes to suggest the next character is longer than the time it takes for you to type.

Monitoring and debugging these models in production is also nontrivial. As ML models get more complex, coupled with the lack of visibility into their work, it's hard to figure out what went wrong or be alerted quickly enough when things go wrong.

The good news is that these engineering challenges are being tackled at a breakneck pace. Back in 2018, when the Bidirectional Encoder Representations from Transformers (BERT) paper first came out, people were talking about how BERT was too big, too complex, and too slow to be practical. The pretrained large BERT model has 340 million parameters and is 1.35 GB. Fast-forward two years later, BERT and its variants were already used in almost every English search on Google.

CONCLUSION

In Artificial Intelligence (AI) / Machine Learning (ML) Systems/Applications, software development, testing and maintenance is just as critical as it is in any other software development form. Due to the nature of how AI/ML Explainability and Transparency: systems work and are built, there are many difficulties that software developers and testers face with AI/ML applications. In the efficient development, testing and maintenance of AI/ML applications, this paper has enumerated and provided an overview of some challenges for future studies, each of these challenges is ideal for seeking solutions that alleviate the problem and illuminate the path to more efficient software testing methods and methodologies that can be applied to AI/ML systems/applications compared to traditional software.

REFERENCES

[1] Fumihiro Kumeno et'al (2019). *Software Engineering Challenges for Machine Learning Applications: A Literature Review*. Sage Journals - First Published online, November 1, 2019. Vol. 13, Issue 4. https://doi.org/10.3233/IDT-190160.

[2] Gorken Giray (2021). A *Software Engineering Perspective on Engineering Machine Learning Systems: State of the Art and Challenges*. ScienceDirect Elsevier Journal of Systems and Software. Vol, 180. October 2021, 111031. https://doi.org/10.1016/j.jss.2021.111031.

[3] Kishore Sugali, Chris Sprunger and Venkata N Inukollu (2021). *Software Testing: Issues and Challenges of Artificial Intelligence & Machine Learning.* International Journal of Artificial Intelligence and Applications (IJAIA), Vol.12, No.1, January 2021 DOI: 10.5121/ijaia.2021.12107 101

[4] Lev Craig (2023). *Compare Machine Learning vs Software Engineering*. TechTarget - O'Reilly Media. Published 16 August, 2023.

[5] Robbie Allen (2020). *How Machine Learning Differs from Traditional Software.* Medium Automated Consulting Group. January 21, 2020.

[6] Silverio Martinez-Fernandez et'al (2022). *Software Engineering for AI-based Systems: A Survey*. ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 31, Issue 2. Article No.: 37e. Pages 1-59, https://doi.org/10.1145/3487043. 01 April, 2022.