

AI-Driven Software Testing: Automating Test Creation and Execution for AI/ML and Real-Time Applications

Anbarasu Arivoli

Company: Target, Minneapolis, MN

Email: [anbarasuarivoli\[at\]gmail.com](mailto:anbarasuarivoli[at]gmail.com)

Abstract: *AI-driven software testing has evolved alongside AI-based and assisted software development. It offers a wide range of benefits for conventional software, but it's critical for more complex applications. This includes AI/ML applications and real-time applications. But in these domains, AI-driven software testing comes with its own set of challenges. This includes the complexity of such applications (AI/ML and real-time), data dependencies, quality issues, the dynamic and evolving nature of these software, scalability and performance concerns, and ethical concerns. In addition to choosing the right AI-driven software testing models that can adapt to the evolving needs of this software, the right approach to AI-driven test case generation, synthetic data generation, and integrating explainability in the development and testing stage can help developers get around this issue.*

Keywords: AI-driven software testing, AI/ML applications, real-time applications

1. Introduction

Artificial Intelligence (AI) has transformed a lot of different industries, including software development. Developers are now using different types of tools, including Large Language Models (LLMs) like OpenAI's ChatGPT and coding assistants like GitHub CoPilot, to develop, optimize, and maintain code. This is true for almost all different types of applications, including, ironically, AI and Machine Learning (ML) applications and real-time applications. Both of these application types have to be more dynamic and adaptable in nature compared to applications dealing with a predictable amount, type, and nature of data. Considering AI is still in its early stages, rigorous and comprehensive testing of sensitive AI/ML applications and real-time applications is needed to ensure that they work as intended. This is where another breed of AI tool comes into play, i. e., AI-driven software testing. Like its influence in development, AI-driven software testing is rapidly growing, and the stakeholders in this industry understand the strengths and limitations of these tools to a healthy extent. However, the challenges associated with AI-driven software testing are enhanced for complex software packages and applications, including AI/ML applications and real-time applications.

2. Literature Review

The literature on this topic has evolved over several years. The early literature on this topic discussed the broader impact of AI on software testing and identified its core strengths, like faster testing (leading to rapid production cycles) [1]. More directed research in software testing for AI/ML systems focused on bridging the gap between traditional methodologies and the demands of intelligent systems [2]. The literature also covered specific test scenarios/software types like cross-platform applications and how AI-driven testing can revolutionize and automate their testing and improve quality [3]. Integrating AI-driven testing into DevOps can significantly enhance and improve the software production lifecycles and allow for smarter, more intelligent

testing, which is critical for a broader range of applications. Early efforts highlighted the limitations of manual test case design in capturing the non-deterministic behavior of ML models, particularly in computer vision and natural language processing [4]. Studies emphasized the need for automated frameworks capable of simulating diverse input distributions and adversarial scenarios to uncover model vulnerabilities [5].

In real-time systems, literature identified the importance of latency-aware testing frameworks to validate performance under strict timing constraints [6]. Researchers explored techniques such as hardware-in-the-loop (HIL) simulations and probabilistic timing analysis to ensure systems met deadlines without compromising accuracy [7].

3. Problem Statements: AI-driven software Testing and Automation Challenges for AI/ML and Real-Time Applications

A wide range of challenges and problems can be associated with AI-driven software testing, particularly for AI/ML applications and a range of real-time applications. This includes but is not limited to:

3.1 AI/ML Application Complexity

Conventional applications, even ones with comprehensive operational scopes, usually have predictable use cases and expectations. Developers have a general idea of how these applications may need to perform under certain conditions, allowing for easier test development. However, many AI/ML applications are designed to handle a wide range of unknown variables and scenarios. This complexity and lack of predictability require testing to be as dynamic as the variability the application is likely to face.

Further complicating the problem is the black-box nature of most AI and ML models and applications based on them.

Volume 12 Issue 7, July 2023

www.ijsr.net

Licensed Under Creative Commons Attribution CC BY

Understanding how a model might behave in certain way is almost impossible to predict so testing scenarios have to be comprehensive in nature as well. There are also hyperparameters, overfitting and underfitting concerns, and other AI/ML related complexities to take into account.

3.2 Real-Time Application Challenges

Real-time application may have their own set of unique challenges different from AI and ML applications, starting with the volume of data they need to handle, fluctuations in data, range, and parameters around the data sets, etc. The most significant challenge domain, however, is rapid real-time processing, response, and decision-making, especially for time-sensitive applications like trading and medical diagnostics. For other applications, the massive amount of data they have to handle and the pre-sorting and cleaning of data within the application can pose a unique set of challenges. So, the testing has to account for not just how well the software/application is performing in relevant scenarios but how rapidly it's performing and how well it acts while different systems are running concurrently. These applications also often have a slew of integrations that have to be tested and handled.

3.3 Data Dependency and Quality Issues

AI models are highly sensitive to input data distribution. Shifts in data—such as seasonal variations in e-commerce traffic or sensor noise in IoT devices—can degrade model performance. Testing frameworks must account for data drift, missing values, and adversarial inputs, which are rarely addressed in conventional testing pipelines. They may also have to test the “corrections” or realignment built into AI/ML and real-time applications (usually outside AI/ML models) to ensure that it's making correct adjustments to outputs and performance of the applications.

3.4 Dynamic and Evolving Models

Many AI systems employ continuous learning, where models update iteratively based on new data. This creates a moving target for testers, as validation suites must adapt to evolving logic without human intervention. A recommendation system that retrains daily, for instance, requires automated regression testing to prevent unintended bias. Testing for such systems and the applications based on them has to evolve alongside the model, too. This may include hyperparameters within the model, the broader scope of the application itself, and its behavior in different scenarios and input cases.

3.5 Scalability and Performance Bottlenecks

As AI applications scale to handle millions of users or devices, testing frameworks must simulate high-load scenarios without prohibitive computational costs. Performance bottlenecks in distributed systems—such as network latency or database contention—are often overlooked during testing.

3.6 Ethical and Security Concerns

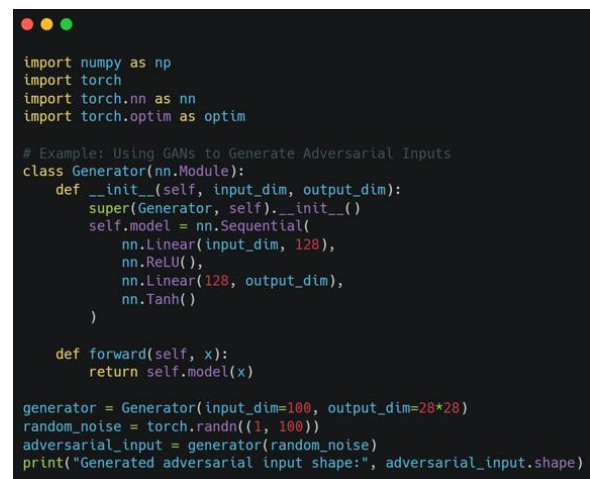
AI systems risk perpetuating biases present in training data or succumbing to adversarial attacks. Testing must validate not only functionality but also fairness, transparency, and resistance to malicious inputs.

4. Solutions: Best AI-Driven Software Testing Practices

Developing the right testing scenarios and using the right automated test creation frameworks can go a long way toward solving these problems.

4.1 AI-Driven Test Case Generation

AI-driven test case generation leverages machine learning algorithms to autonomously identify high-risk scenarios and optimize validation processes. Traditional methods, which rely on predefined scripts, struggle to anticipate the unpredictable behavior of AI/ML systems. Generative Adversarial Networks (GANs) address this by synthesizing edge cases—such as adversarial images with imperceptible noise or sensor data simulating rare hardware failures—that expose model vulnerabilities. These synthetic inputs force models to confront scenarios absent from training data, revealing biases or overfitting. Reinforcement Learning (RL) further enhances test coverage by enabling agents to explore the system's decision space dynamically. RL agents learn to prioritize test cases that maximize defect discovery, such as inputs that trigger conflicting predictions in autonomous systems. Meanwhile, metamorphic testing infers relationships between input transformations and expected outcomes (e. g., rotating an image should not alter its classification label) to validate consistency. This triad of approaches—GANs, RL, and metamorphic testing—enables frameworks to adaptively probe AI systems, ensuring robustness against both known and unforeseen failure modes.



```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim

# Example: Using GANs to Generate Adversarial Inputs
class Generator(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, output_dim),
            nn.Tanh()
        )

    def forward(self, x):
        return self.model(x)

generator = Generator(input_dim=100, output_dim=28*28)
random_noise = torch.randn((1, 100))
adversarial_input = generator(random_noise)
print("Generated adversarial input shape:", adversarial_input.shape)
```

Figure 1: Adversarial Inputs Using GAN

4.2 Adaptive Testing Frameworks for Evolving Models

Continuous learning models, which update iteratively in response to new data, demand testing frameworks capable of evolving in tandem. Adaptive testing integrates with CI/CD pipelines to automate validation across model versions. Dynamic regression testing employs ML algorithms to compare predictions between iterations, flagging deviations

that indicate performance regressions or unintended behavioral shifts. For instance, a recommendation system retrained daily might inadvertently prioritize controversial content; automated comparisons against baseline metrics can detect such anomalies. Concept drift detection complements this by monitoring input data distributions in real-time. Statistical tests, such as Kolmogorov-Smirnov or chi-squared analyses, quantify deviations from expected data patterns. When drift exceeds predefined thresholds—such as sudden changes in user demographics or sensor calibration—the framework triggers retraining or retesting. The goal of these measures is to ensure that the underlying models remain aligned with operational environments, even as the data changes and evolves over time.

```
#Example: Detecting Concept Drift Using Kolmogorov-Smirnov Test
from scipy.stats import ks_2samp

def detect_concept_drift(old_data, new_data, threshold=0.05):
    stat, p_value = ks_2samp(old_data, new_data)
    if p_value < threshold:
        print("Concept drift detected!")
    else:
        print("No significant drift detected.")

# Simulated data distributions
old_data = np.random.normal(0, 1, 1000)
new_data = np.random.normal(0.5, 1, 1000)
detect_concept_drift(old_data, new_data)
```

Figure 2: Detecting Concept Drift

4.3 Anomaly Detection and Monitoring in Real-Time

Real-time systems, which include algorithmic trading platforms and autonomous drones, often require testing frameworks that validate not just functional correctness but timing constraints as well. AI-powered anomaly detection tools process telemetry streams to identify latency spikes, resource contention, or prediction outliers. Stream processing engines like Apache Flink or Kafka Streams analyze high-velocity data, applying unsupervised learning models to detect anomalies in execution times or throughput. For example, a medical monitoring system might use clustering algorithms to flag irregular heart rate patterns that deviate from established norms. Predictive latency modeling augments this by forecasting execution times under varying loads. Regression models trained on historical performance data predict how system latency scales with input volume, enabling preemptive optimization. This extends to ensuring that applications meet Service-Level Agreements (SLAs) constraints and requirements, which can often be quite demanding.

4.4 Synthetic Data Generation (including its Augmentation)

One of the main problems AI testing and AI development face is the lack of the right type of data. One way to solve this issue is to use AI models to generate the type of data needed to train and test certain AI models. Another problem is that when there is enough data on hand, it's either biased in some regard or over-presents or underrepresents certain elements within a population. Healthcare data in the US is mostly comprised of a single race, underrepresenting the needs of other races. The right approach to AI-generated data, i. e., synthetic data, can resolve this issue. These models/systems can be used to

generate data that mimic the underlying realities of the real world without bias or malice. There are different types of synthetic data generators that are useful for different types of requirements.

This includes Variational Autoencoders (VAEs) and diffusion models that are capable of producing high-quality and realistic simulations of rare events, such as equipment malfunctions in industrial IoT systems or atypical symptoms in medical diagnostics. The datasets for these scenarios are either too small or too biased in the real world. In facial recognition systems, synthetic data can represent underrepresented demographics, reducing racial or gender bias. Augmentation techniques further diversify training data by applying transformations like noise injection or geometric distortions. Bias mitigation algorithms, such as reweighting or adversarial debiasing, adjust training data distributions to ensure fairness. For instance, a loan approval model might be tested for demographic parity by comparing approval rates across synthetic datasets with balanced ethnic representation.

4.5 Explainability and Transparency Tools

Explainable AI (XAI) techniques integrate with testing frameworks to audit model decisions and enhance trust. Saliency maps visualize input features influencing predictions, enabling testers to identify overreliance on spurious correlations. In a computer vision model, saliency maps might reveal that a tumor detector focuses on image artifacts rather than biological structures. Counterfactual analysis generates minimal perturbations to inputs—such as altering a single pixel in an image—to test model sensitivity. These “what-if” scenarios isolate failure modes and validate decision boundaries. These tools also ensure compliance with regulations like the EU AI Act, which mandates transparency in high-risk systems.

4.6 Ethical and Security Validation

AI testing frameworks must validate ethical compliance and resilience against adversarial attacks. Robustness testing subjects models to adversarial inputs, such as perturbed audio commands that mislead voice assistants or adversarial patches that confuse object detectors. Penetration testing simulates attacks like model inversion, where adversaries extract training data from API responses. Fairness testing employs metrics like equalized odds or demographic parity to quantify disparities in outcomes across groups. For example, a hiring tool might be tested for gender bias by evaluating its shortlisting rates against synthetic resumes with randomized genders. These practices ensure systems align with ethical guidelines and withstand malicious exploitation.

5. Conclusion

AI-driven software testing, whether it's automated for specific scenarios or evolving to meet the changing needs of an AI/ML system, is paradigm-shifting, to say the least. The right AI-powered adaptive testing frameworks and models can make the process of testing not just smooth but also much more comprehensive and useful than manual testing could ever be. Its benefits of freeing up development teams from testing responsibility is one of the most cited and easily understood

ones. But it's worth understanding that in many scenarios, it's AI-driven software testing is not an added benefit. It's critical to test AI/ML applications and real-time applications that, thanks to their complexity, cannot be tested manually.

References

- [1] "The Impact of Artificial Intelligence on Software Testing." Accessed: Mar.05, 2025. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8717439>
- [2] V. Jeremić, R. Bucea-Manea-Tonis, S. Vesić, and H. Stefanović, "Revolutionizing Software Testing: The Impact of AI, ML, and IoT".
- [3] N. Srinivas, N. Mandalaju, and S. V. Nadimpalli, "Cross-Platform Application Testing: AI-Driven Automation Strategies," *Artif. Intell. Mach. Learn. Rev.*, vol.1, no.1, Art. no.1, Jan.2020, doi: 10.69987/AIMLR.2020.10102.
- [4] T. M. King, J. Arbon, D. Santiago, D. Adamo, W. Chin, and R. Shanmugam, "AI for Testing Today and Tomorrow: Industry Perspectives, " in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, Apr.2019, pp.81–88. doi: 10.1109/AITest.2019.000-3.
- [5] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated Whitebox Testing of Deep Learning Systems, " in *Proceedings of the 26th Symposium on Operating Systems Principles*, in SOSP '17. New York, NY, USA: Association for Computing Machinery, Oct.2017, pp.1–18. doi: 10.1145/3132747.3132785.
- [6] R. Jabbar, M. Krichen, M. Shinoy, M. Kharbeche, N. Fetais, and K. Barkaoui, "A Model-Based and Resource-Aware Testing Framework for Parking System Payment using Blockchain, " in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, Jun.2020, pp.1252–1259. doi: 10.1109/IWCMC48107.2020.9148212.
- [7] S. Raikwar, L. Jijyabhau Wani, S. Arun Kumar, and M. Sreenivasulu Rao, "Hardware-in-the-Loop test automation of embedded systems for agricultural tractors," *Measurement*, vol.133, pp.271–280, Feb.2019, doi: 10.1016/j.measurement.2018.10.014.