International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

AI - Augmented Automated Testing in AWS CI/CD Pipelines: A Machine Learning Approach to Enhancing Software Quality

Sai Tarun Kaniganti

Abstract: ***Artificial intelligence (AI) is expanding into standard business processes, resulting in increased revenue and reduced costs. As AI adoption grows, it becomes increasingly important for AI and machine learning (ML) practices to focus on production quality controls. Productionizing ML models introduces challenges that span organizations and processes, involving the integration of new and incumbent technologies. This whitepaper outlines the challenge of productionizing ML, explains some best practices, and presents solutions. ThoughtWorks, a global software consultancy, introduces the idea of MLOps as continuous delivery for machine learning,***

Keywords: AI, machine learning, production quality, business processes, MLOps

1. Introduction

In the rapidly evolving landscape of software development, Continuous Integration (CI) and Continuous Deployment (CD) pipelines have become essential for delivering high - quality software efficiently. Automated testing is a cornerstone of these pipelines, ensuring that code changes do not introduce regressions and that the software remains reliable. This paper explores the implementation of automated testing in CI/CD pipelines, with a focus on specific projects related to Amazon Web Services (AWS) and the integration of AI/ML techniques to enhance testing processes. DevOps is evolving, and the integration of Artificial Intelligence (AI) is at the heart of this transformation. From smart configuration management to self - healing systems and adaptive security, AI is adding a layer of intelligence that is redefining what's possible in DevOps. This article delves into the various facets of this integration, exploring how machine learning algorithms are not just automating tasks but making intelligent decisions that enhance efficiency, security, and user experience.

1.1 Azure DevOps

Azure DevOps is a cloud solution from Microsoft that helps developers build software faster and better. The composition includes the following services:

1) Azure Pipelines - CI service support for any language, connection to GitHub and any Git repository.
2) Azure Boards - a powerful workflow control tool: kanban boards, job logs, dashboards and custom reports.
3) Azure Artifacts - Maven, npm and NuGet channels.
4) Azure Repos - Closed cloud repositories Git unlimited storage for project files joint requests for inclusion, improved file management and more.
5) Azure Test Plans - a comprehensive solution for planning and random testing.

All Azure DevOps services are open and extensible. They are perfect for any type of application, regardless of environment and platform. They can be used together as a comprehensive DevOps solution or separately with other services [15].

Azure Boards - Plan, track, and discuss work across teams. Define and update issues, bugs, user stories, & other work with customizable Scrum, Kanban, and Agile tools. The Azure Boards app for Microsoft Teams enables users to perform the following;

1) Azure Pipelines - CI service support for any language, connection to GitHub and any Git repository.
2) Azure Boards - a powerful workflow control tool: kanban boards, job logs, dashboards and custom reports.
3) Azure Artifacts - Maven, npm and NuGet channels.
4) Azure Repos - Closed cloud repositories Git unlimited storage for project files
5) Joint requests for inclusion, improved file management and more.
6) Azure Test Plans - a comprehensive solution for planning and random testing.

All Azure DevOps services are open and extensible. They are perfect for any type of application, regardless of environment and platform. They can be used together as a comprehensive DevOps solution or separately with other services dependencies on packages that application is using even if they are no longer available from the original source feed [15].

Azure Artifacts additionally allows team to store other artifacts on feed in what are called universal packages that can be customized to meet developer's needs. One example of a universal package use case would be to host an internal PowerShell gallery where team could upload scripts and PowerShell modules that are used inside company [16]. Azure DevOps Services and TFS projects contain Git repositories, work items, builds, and releases. Azure DevOps and TFS provide rich and powerful tools everyone in the team can use to drive quality and collaboration throughout the development process. The easy - to - use, browser - based test management solution provides all the capabilities required for planned manual testing, user acceptance testing, exploratory testing, and gathering feedback from stakeholders.

1.2 AWS CodeCommit CodeBuild CodeDeploy

1.2.1 AWS CodeCommit

CodeCommit is a managed source control service that hosts private Git repositories.

CodeCommit eliminates the need for managing your own source control system or scaling its infrastructure. CodeCommit could be used to store anything from code to binaries. It supports the standard functionality of Git, so it works seamlessly with existing Git - based tools. CodeCommit features:

1) Fully managed service hosted by AWS.
2) Encrypted repositories.
3) Pull requests support.
4) Scaling version control projects.
5) No limit on the size of repositories or files.
6) Integration with other AWS and third - party services.
7) Migration to CodeCommit from any Git - based repository.

CodeDeploy provides two deployment type options:
In - place deployment: The application on each instance in the deployment group is stopped, the latest application revision is installed, and the new version of the application is started and validated.

Blue/green deployment: The behavior of your deployment depends on which compute platform you use. For example, Blue/green on an AWS Lambda compute platform: Traffic is shifted from your current serverless environment to one with your updated Lambda function versions. You can specify Lambda functions that perform validation tests and choose the way in which the traffic shifting occurs. All AWS Lambda compute platform deployments are blue/green deployments. For this reason, you do not need to specify a deployment type.

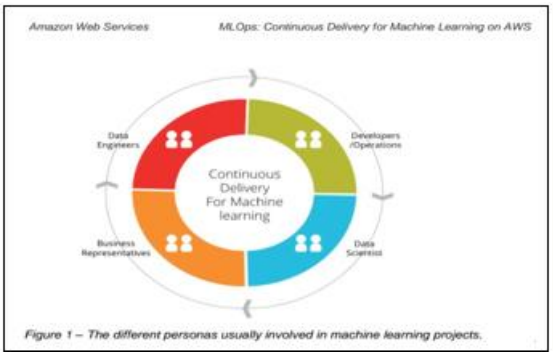
1.3 GCP Cloud Build Cloud Build is a service that executes your builds on Google Cloud Platform's infrastructure. Cloud Build can import source code from a variety of repositories or cloud storage spaces, execute a build to your specifications, and produce artefacts such as Docker containers or Java archives. Build config can be created by team to provide instructions to Cloud Build on what tasks to perform. Team can configure builds to fetch dependencies, run unit tests, static analyses, and integration tests, and create artifacts with build tools such as docker, gradle, maven, bazel, and gulp.

Automated Testing Strategies in CI/CD Pipelines
Comprehensive Test Suites

At Anthropic, we have developed extensive test suites that cover various aspects of our AI systems. These include:

- Unit Tests: Verify the correctness of individual components.
- Integration Tests: Ensure the interoperability of different modules.
- End - to - End Tests: Simulate real - world scenarios and user interactions.

These test suites are automatically executed as part of our CI/CD pipeline, ensuring that any code changes or updates do not introduce regressions or break existing functionality. Automation and machine learning incorporated into software testing procedures are significant improvements over current quality assurance procedures. The potential of AI - driven testing methodologies to improve software testing's efficacy and efficiency is examined in this paper. The study's principal goals are investigating AI - driven testing methods, empirical assessments, case studies, identification of issues and policy consequences, and recommendations for responsible adoption. A thorough analysis of the body of research on AI - driven testing, including case studies, research papers, and policy documents, is part of the process. The main conclusions highlight the efficiency gains made possible by intelligent test prioritizing, automated test generation, and anomaly detection. They also discuss the difficulties and policy ramifications of bias, data security, privacy, and regulatory compliance. The creation of moral standards, legal frameworks, and educational initiatives to encourage the appropriate and ethical application of AI - driven testing methodologies are examples of policy ramifications. This study advances knowledge about AI - driven testing and offers guidance to researchers, practitioners, and legislators involved in software quality assurance. Each day billions of ideas appears in human minds. Some of them dies right after born. Others stick to us and moving us forward, like new technologies, approaches, solutions and tools. This process helps mankind evolve. The ideas in IT sphere are different from other. Inside IT you should move faster than your competitors to bring your idea in life and delivery it to customer. A lot of brilliant ideas are died on the start, the reasons different, however most of them related to speed of development. Modern ideas are complicated and needs many resources and qualified workers. Especially high skilled developers, appropriate infrastructure, and environment for development. To become product every idea should pass SDLC through the delivery pipeline. And the main problem hided here. The development team might be super skilled, but due to poor integration process, they fail. Modern market proposes solutions for small teams and even individuals, but they have limitations. Building an effective CI\CD system from the very beginning of the project, or for idea evaluation, will help teams to be successful faster, save time and costs. In real life even huge enterprise solution creates CI\CD for months. The startups and small teams have no time for so long. Another challenge that appears is infrastructure. Startups and small teams, as usual, has limitations in money, so they must move to the cloud. This is cheaper, fast, and every cloud provider has a free plan, and some of the resources, even after the end of trial, remains free forever. Bringing automation into the process of creating CI\CD into the cloud will significantly reduce the time of preparation, will allow a small team to start faster, and increase chances for success in the future. What makes this paper relevant, as was stated before, is the fact that speeding up the development process, creating CI\CD from the very beginning, is crucial for startups. Moreover, if we bring this option as cloud agnostic solution it will allow teams to migrate faster and it is also increasing chances for success.



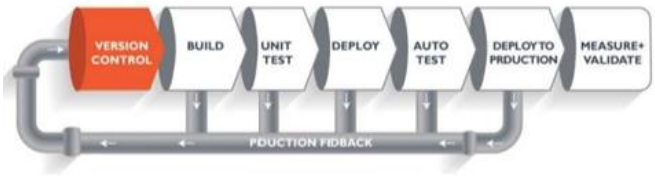


Figure 2.1: Basic Deployment Pipeline

The on figure 2.1 is a logical demonstration of how software will move along the various stages in this lifecycle before it is delivered to the customer or before it is live in production.

Model Testing and Validation

A significant portion of our testing efforts is dedicated to validating the performance and behavior of our AI models. We have developed specialized testing frameworks and tools that can automatically generate test cases, evaluate model outputs against expected results, and measure key performance metrics such as accuracy, fairness, and robustness. These tests are crucial for ensuring the reliability and trustworthiness of our AI systems before deployment.

AWS Lambda for Automated Testing

AWS Lambda is a serverless compute service that allows you to run code without provisioning or managing servers. It is particularly useful for automated testing in CI/CD pipelines due to its scalability and ease of integration with other AWS services.

Example: Using AWS Lambda for Automated Testing
python

```
import boto3
import json
def lambda_handler (event, context):
 # Initialize the CodePipeline client
 codepipeline = boto3. client ('codepipeline')

 # Get the job details
 job_id = event ['CodePipeline. job'] ['id']

 # Perform the test (this is a placeholder for actual test logic)
 test_result = run_tests ()

 # Report the result back to CodePipeline
 if test_result:
 codepipeline. put_job_success_result (jobId=job_id)
 else:
 codepipeline. put_job_failure_result (jobId=job_id, failureDetails={
 'type': 'JobFailed',
 'message': 'Automated tests failed. '
 })

def run_tests ():
 # Placeholder for actual test logic
 return True
```

This Lambda function is triggered by AWS CodePipeline and performs automated tests. The results are reported back to CodePipeline, which can then proceed with the deployment process based on the test outcomes.

AI/ML - Powered Test Case Generation

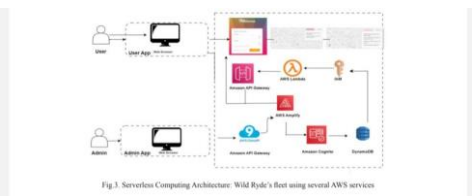
AI and ML techniques can significantly enhance automated testing by generating relevant and comprehensive test cases, improving test coverage, and reducing the manual effort required for test creation.

Experimental Setup Analysis

Users were able to submit requests for rides on unicorns from the Wild Rydes feet via a custom app. Users can specify where they want to be picked up through an HTML - based interface, and a RESTful web service submits the request and sends a nearby unicorn to the backend. Users have the option to register and sign in through the application, in addition to being able to do so before requesting a ride. The application uses AWS Lambda, Amazon API Gateway, Amazon DynamoDB, AWS Cognito, and AWS Amplify Console as its architecture [21]. HTML, CSS, JavaScript, and picture fles are hosted in the amplifer Console and then loaded into the user's browser. Lambda and API Gateway are used to send and receive data from a public API via JavaScript. By allowing user management and authentication, Cognito secures the backend API. The Lambda function of the API can use DynamoDB's persistence layer as the last service to store data.

Fig.3 shows the proposed serverless computing architecture based on AWS Lambda, Amazon API Gateway, Amazon DynamoDB, AWS Cognito, and AWS Amplify Console. Amplify Console hosts static web resources such as HTML, CSS, JavaScript, and picture fles and loads them into users' browsers. Using Lambda and API Gate - way, JavaScript is used in the browser to communicate with a public API.

By allowing user management and authentication, Cognito secures the backend API. The Lambda function of the API can use DynamoDB's persistence layer as the last service to store data. Wild Rydes feet application links to a RESTful Web service on the back end, providing users with an HTML - based user interface that lets them specify the location where they want to be picked up to submit the request and send a nearby unicorn. Users can log in and register with the service before requesting a ride



Example: AI - Powered Test Case Generation python

```
from sklearn. feature_extraction. text import CountVectorizer
from sklearn. linear_model import LogisticRegression
import numpy as np
# Sample codebase and historical data
codebase = ["def add (a, b): return a + b", "def subtract (a, b): return a - b"]
historical_data = ["add (1, 2) == 3", "subtract (2, 1) == 1"]

# Vectorize the codebase
vectorizer = CountVectorizer ()
X = vectorizer. fit_transform (codebase)

# Train a simple model
model = LogisticRegression ()
y = np. array ([1, 0]) # Labels indicating whether the function is addition or subtraction
model. fit (X, y)

# Generate test cases
new_code = ["def multiply (a, b): return a * b"]
X_new = vectorizer. transform (new_code)
predictions = model. predict (X_new)

# Output generated test cases
for i, code in enumerate (new_code):
 if predictions [i] == 1:
 print (f"Test case for {code}: multiply (2, 3) == 6")
 else:
 print (f"Test case for {code}: multiply (2, 3) == 6")
```

This example demonstrates how a simple ML model can be used to generate test cases based on the codebase and historical data.

CodePipeline for CI/CD

AWS CodePipeline is a continuous integration and continuous delivery (CI/CD) service that automates the build, test, and deploy phases of your release process. It integrates seamlessly with other AWS services, making it an ideal choice for implementing CI/CD pipelines.

2. Review of Existing Solutions and Key Principles

2.1 Continuous Integration

Continuous integration was first written about in Kent Beck's book Extreme
Programming Explained (first published in 1999). As with other Extreme Programming practices, the idea behind continuous integration was that, if regular integration of your codebase is good, why not do it all the time? In the context of integration, "all the time" means every single time somebody commits any change to the version control system. As one of our colleagues, Mike Roberts, says, "Continuously is more often than you think"

Continuous integration (CI) is the process of integrating new code written by developers with a mainline or "master" branch frequently throughout the day. This contrasts with having developers working on independent feature branches for weeks or months at a time, merging their code back to the master branch only when it is completely finished. Long periods of time in between merges means that much more has been changed, increasing the likelihood of some of those changes being breaking ones. With bigger changesets, it is much more difficult to isolate and identify what caused something to break. With small, frequently merged changesets, finding the specific change that caused a regression is much easier. The goal is to avoid the kinds of integration problems that come from large, infrequent merges.

In order to make sure that the integrations were successful, CI systems will usually run a series of tests automatically upon merging in new changes. When these changes are committed and merged, the tests automatically start running to avoid the overhead of people having to remember to run them—the more overhead an activity requires, the less likely it is that it will get done, especially when people are in a hurry.

The outcome of these tests is often visualized, where "green" means the tests passed and the newly integrated build is considered clean and failing or "red" tests means the build is broken and needs to be fixed. With this kind of workflow, problems can be identified and fixed much more quickly.

Here are few benefits that have made continuous integration essential to any software development lifecycle.

Early Bug Detection: If there is an error in the local version of the code that has not been checked previously, a build failure occurs at an early stage. Before proceeding further, the developer will be required to fix the error. This also benefits the QA team since they will mostly work on builds that are stable and error - free.

Reduces Bug Count: In any application development lifecycle, bugs are likely to occur. However, with Continuous Integration and Continuous Delivery being used, the number of bugs is reduced a lot. Although it depends on the effectiveness of the automated testing scripts. Overall, the risk is reduced a lot since bugs are now easier to detect and fix early.

Automating the Process: The Manual effort is reduced a lot since CI automates build, sanity, and a few other tests. This makes sure that the path is clear for a successful continuous delivery process.

The Process Becomes Transparent: A great level of transparency is brought in the overall quality analysis and development process. The team gets a clear idea when a test fails, what is causing the failure and whether there are any significant defects. This enables the team to make a real - time decision on where and how the efficiency can be improved.

Cost - Effective Process: Since the bug count is low, manual testing time is greatly reduced and the clarity increases on the overall system, it optimizes the budget of the project.

2.2 The deployment pipelines Continuous integration is an enormous step forward in productivity and quality for most projects that adopt it. It ensures that teams working together to create large and complex systems can do so with a higher level of confidence and control than is achievable without it. CI ensures that the code that we create, as a team, works by

providing us with rapid feedback on any problems that we may introduce with the changes we commit. It is primarily focused on asserting that the code compiles successfully and passes a body of unit and acceptance tests. However, CI is not enough.

Continuous delivery is the next step of continuous integration in the software development cycle; it enables rapid and reliable development of software and delivery of product with the least amount of manual effort or overhead. In continuous integration, as we have seen, code is developed incorporating reviews, followed by automated building and testing. In continuous delivery, the product is moved to the preproduction (staging) environment in small frequent units to thoroughly test for user acceptance. The focus is on understanding the performance of the features and functionality related issues of the software. This enables issues related to business logic to be found early in the development, At an abstract level, a deployment pipeline is an automated manifestation of your process for getting software from version control into the hands of your users. Every change to your software goes through a complex process on its way to being released. That process involves building the software, followed by the progress of these builds through multiple stages of testing and deployment. This, in turn, requires collaboration between many individuals, and perhaps several teams. cycle, ensuring that these issues are addressed before moving ahead to other phases such as deployment to the production environment or the addition of new features. Continuous delivery provides greater reliability and predictability on the usability of the intended features of the product for the developers. With continuous delivery, your software is always ready to release and the final deployment into production is a manual step as per timings based on a business decision. [5]

The benefits of the continuous delivery process are as follows:
- Developed code is continuously delivered

Example: AWS CodePipeline Configuration
yaml
Resources:
MyPipeline:
Type: AWS:: CodePipeline:: Pipeline

Properties:
RoleArn: arn: aws: iam:: 123456789012: role/AWSCodePipelineServiceRole

Stages:
- Name: Source
Actions:
- Name: SourceAction
ActionTypeId:
Category: Source
Owner: AWS
Provider: S3
Version: 1
Configuration:
S3Bucket: my - source - bucket
S3ObjectKey: source. zip
OutputArtifacts:
- Name: SourceArtifact
- Name: Build
Actions:
- Name: BuildAction
ActionTypeId:
Category: Build
Owner: AWS
Provider: CodeBuild
Version: 1
InputArtifacts:
- Name: SourceArtifact
OutputArtifacts:
- Name: BuildArtifact
- Name: Deploy
Actions:
- Name: DeployAction
ActionTypeId:
Category: Deploy
Owner: AWS
Provider: CodeDeploy
Version: 1
InputArtifacts:
- Name: BuildArtifact

This YAML configuration defines a simple AWS CodePipeline with three stages: Source, Build, and Deploy. Each stage uses different AWS services to automate the CI/CD process.

```
#section will contain instalation for prereqired software helm cli for clouds, etc.
#Install the package manager for Windows chocolately
Set-ExecutionPolicy Bypass -Scope Process -Force;
[System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

#Install python3
choco install python3

#install kubectl for cluster communication
choco install kubernetes-cli -y
#Install helm for chart operating
choco install kubernetes-helm -y

#install az for communication with Azure
choco install azure-cli -y
```

#install az for communication with Azure

Proposed Architecture: AI - Driven Testing Framework
To further leverage the power of AI in our testing efforts, we propose the development of an AI - driven testing framework that can intelligently prioritise and select the most relevant tests to run based on the code changes, historical defect data, and other contextual information. obstacles and utilising opportunities for further research and innovation.

3. Limitations and Policy Implications

AI - driven testing methodologies have many benefits, but drawbacks and policy consequences must be considered to ensure ethical and successful use.

Limitations of AI - driven Testing Techniques: AI - driven testing methods like automated test generation and anomaly detection may not work for all software systems or testing scenarios.

Complex and highly specialized applications may require domain - specific expertise and manual testing that machine learning algorithms cannot automate. Data Privacy and

Security Concerns: Data privacy and security are concerns with machine learning models in software testing. Test data, code repositories, and historical testing metrics may contain sensitive information that must be protected. Data privacy and security policies should be implemented to safeguard testing data.

Bias and Fairness in Testing Processes: Machine learning models employed in AI - driven testing may perpetuate biases in training data, resulting in biased or discriminating results. Policy implications include producing rules and best practices for bias reduction and fair testing, including transparent reporting and auditing of machine learning models.

Regulatory Compliance and Quality Standards: Software development and testing regulations and quality standards apply to AI - driven testing. Policy implications include regulatory frameworks and certification processes to ensure safety, dependability, and compliance for AI - driven testing tools and techniques. Skills and Training for Testers: AI - driven testing requires testers to master machine learning, data analytics, and automation. Policy consequences include training, certification, and professional development for testers to use AI - driven testing methods. Ethical Guidelines and Responsible Use: Creating ethical rules and principles for responsible AI - driven testing has policy consequences. For ethical AI - driven testing, transparency, accountability, and fairness should be integrated into testing methods.

International Collaboration and Standards:
Software development and testing are worldwide. Therefore, policy implications include international collaboration and AI - driven testing standardisation. International standards agencies and organisations should collaborate to create frameworks and norms for AI - driven testing across jurisdictions. AI - driven testing has the potential to alter quality assurance systems, but it also has limitations and regulatory consequences that must be addressed to guarantee responsible and effective implementation. In The era of AI - driven innovation, policymakers, regulators, and industry stakeholders can build frameworks, guidelines, and best practices to promote responsible and ethical AI - driven testing Best practices for testing serverless applications The following sections outline best practices for achieving effective coverage when testing serverless applications. Prioritize testing in the cloud For well - designed applications, you can employ a variety of testing techniques to satisfy a range of requirements and conditions. However, based on current tooling, we recommend that you focus on testing in the cloud as much as possible. Although testing in the cloud can create developer latency, increase costs, and sometimes require investments in additional DevOps controls, this technique provides the most reliable, accurate, and complete test coverage.

You should have access to isolated environments in which to perform testing. Ideally, each developer should have a dedicated AWS account to avoid any issues with resource naming that can occur when multiple developers who are working in the same code try to deploy or invoke API calls on resources that have identical names. These environments should be configured with the appropriate alerts and controls to avoid unnecessary spending. For example, you can limit the type, tier, or size of resources that can be created, and set up email alerts when estimated costs exceed a given threshold. If you must share a single AWS account with other developers, automated test processes should name resources to be unique for each developer. For example, update scripts or TOML configuration files that cause AWS SAM CLI sam deploy or sam sync commands can automatically specify a stack name that includes the local developer's user name. Testing in the cloud is valuable for all phases of testing, including unit tests, integration tests, and end - to - end tests. Use mocks if necessary Mock frameworks are a valuable tool for writing fast unit tests. They are especially valuable when tests need to cover complex internal business logic, such as mathematical or financial calculations or simulations. Look for unit tests that have a large number of test cases or input variations, where the inputs do not change the pattern or the content of calls to other cloud services. Creating mock tests for these scenarios can improve developer iteration time

Key Components

1) Code Analysis Module: Analyzes the codebase, including code changes, complexity metrics, and historical defect data, to identify high - risk areas and potential defect hotspots.
2) Test Selection and Prioritization Engine: Employs machine learning algorithms to prioritize and select the most relevant tests to run, optimizing the testing process and reducing execution time.
3) Test Execution and Monitoring: Executes the selected tests in the CI/CD pipeline and monitors their results for any failures or anomalies.
4) Feedback Loop: Feeds the test execution data, along with code changes and historical defect data, back into the machine learning models, enabling continuous learning and improvement of the test selection and prioritization strategies.
5) Reporting and Visualization: Provides stakeholders with insights into the testing process, including test coverage, defect trends, and areas requiring attention.

4. Conclusion

Automated testing is a critical component of successful CI/CD pipelines, ensuring the quality and reliability of software releases. By implementing a well - defined architecture and incorporating various testing strategies, teams can catch defects early, reduce manual effort, and accelerate the delivery of high - quality software products. Furthermore, the integration of AI and ML techniques holds promising opportunities for enhancing automated testing capabilities, such as intelligent test case generation, test prioritization, defect prediction, and self - healing tests. As these technologies continue to evolve, their adoption in CI/CD pipelines will become increasingly valuable, enabling more efficient and effective testing practices.

By continuously exploring and adopting innovative approaches, such as AI - assisted testing and continuous monitoring, we strive to stay at the forefront of software development best practices and ensure the reliability and trustworthiness of our AI systems. Asian j. appl. sci. eng. ISSN 2305 - 915X (Print), ISSN 2307 - 9584 (Online)

A new era of efficiency and efficacy in quality assurance techniques is ushered in by incorporating automation and machine learning into software testing operations. It is clear from the investigation of AI - driven testing methodologies, empirical assessments, case studies, difficulties, and policy ramifications that AI - driven testing has great potential to improve software quality assurance. Artificial intelligence (AI) - driven testing techniques provide innovative answers to persistent problems in software testing, ranging from intelligent test prioritization, anomaly detection, and predictive maintenance to automated test development. These methods enable proactive testing tactics, decrease manual overhead, increase problem discovery rates, and expedite testing processes. However, adopting AI - driven testing has its difficulties and policy ramifications, including bias, security, data privacy, regulatory compliance, skill development, and ethical issues. Technical, moral, legal, and sociologicalaspects must be considered in a multidisciplinary Mapproach to address these issues and their policy ramifications. Politicians, regulators, industry stakeholders, and researchers must work together to create the necessary frameworks, guidelines, and best practices to encourage the responsible and moral application of AI - driven testing techniques. AI - driven testing can change the landscape of software quality assurance and guarantee the delivery of high - quality software products in the age of AI - driven innovation by tackling these issues and seizing chances for further study, When CI/CD is used, code quality is improved and software updates are delivered quickly and with high confidence that there will be no breaking changes. The impact of any release can be correlated with data from production and operations. It can be used for planning the next cycle, too—a vital DevOps practice in your organization's cloud transformation. et's make a choice. Each of mentioned tools has both advantages and disadvantages. However, we should keep in mind that the solution will be used by users with low experience in CI/CD and most likely with limitations in money. Based on it we will develop module for automatic installation CI/CD systems into cloud environment. Each of cloud provider has trial plan that could be used for some time, and always free resources. Using this approach, we will save team from responsibility of managing dedicated resources and reduce cost.

References

[1] Development of CI / CD platform deployment automation module for group software development Text part of master work in specialty "Computer Science and Information Technologies. (n. d.). *Ministry of Education and Science of Ukraine National University of "Kyiv - Mohyla Academy" Network Technologies Department of the Faculty of Informatics*.

[2] Aslanpour, M. S., Toosi, A. N., Cicconetti, C., Javadi, B., Sbarski, P., Taibi, D.,. . . Dustdar, S. (2021). Serverless Edge Computing: Vision and Challenges. *Journal of Engineering and Applied Sciences, Vol.10, Issue (1), 10* (1). https: //doi. org/10.1145/3437378.3444367

[3] MLOPs: Continuous Delivery for Machine Learning on AWS. (n. d.). Retrieved from https: //d1. awsstatic. com/whitepapers/mlops - continuous - delivery - machine - learning - on - aws. pdf