



T.C.
ÜSKÜDAR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ YÜKSEK LİSANS PROGRAMI
YÜKSEK LİSANS TEZİ

**YAPAY ZEKA TEKNİKLERİ KULLANARAK KOD TESTİNİ
GELİŞTİRMEK: UYGULAMALI VE ANALİTİK BİR ÇALIŞMA**

Ahmed RAMADAN

Tez Danışmanı
Dr. Husam YASIN

İSTANBUL - 2024

T.C.
ÜSKÜDAR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ YÜKSEK LİSANS PROGRAMI
YÜKSEK LİSANS TEZİ

YAPAY ZEKA TEKNİKLERİ KULLANARAK KOD TESTİNİ
GELİŞTİRMEK: UYGULAMALI VE ANALİTİK BİR ÇALIŞMA

Ahmed RAMADAN

Tez Danışmanı
Dr. Husam YASIN

İSTANBUL-2024

ÖZET

Yapay Zeka Teknikleri Kullanarak Kod Testini Geliştirmek: Uygulamalı ve Analitik Bir Çalışma

Yazılım testi, yazılımın kalitesini ve güvenilirliğini sağlamada çok önemli bir rol oynar. Ancak geleneksel test yöntemleri, yazılım sistemlerinin artan karmaşıklığına ve gelişen doğasına ayak uydurmakta zorluk çekiyor. Yapay zeka (AI) ve makine öğrenimi (ML), kod testi verimliliğini ve etkinliğini artırmak için umut verici yollar sunar.

Bu analitik çalışma, yazılım test uygulamalarını iyileştirmede AI tekniklerinin potansiyelini araştırıyor. AI ve ML'nin temel kavramlarına kapsamlı bir genel bakışın yanı sıra, AI destekli yazılımların test edilmesiyle ilgili belirli zorluklar sağlar. Bu araştırma, yapay zekanın kod testi yöntemini nasıl geliştirebileceğini araştırıyor. Ayrıca, çeşitli ML yaklaşımlarının yazılım test sürecine nasıl entegre edilebileceğini inceleyerek bu alandaki yerleşik araştırmalardan içgörüler elde ediyor. Ayrıca makine öğreniminin modern test araçları üzerindeki etkisini de analiz ederek hem ticari hem de açık kaynak seçeneklerini değerlendiriyor.

İşte araştırmanın bulguları:

Bölüm 1: Temel Bilgiler - Bu bölümde AI ve ML'nin ne olduğu ve AI kullanan yazılımları test etmenin özel zorlukları açıklanmaktadır.

Bölüm 2: AI Çerçevelerini Anlamak - Bu bölümde, yazılımı yapay zeka ile test etmenin iki farklı yolunu temsil eder: Yazılım Testi Derneği için AI modeli ve Paul Gerrard'ın iyi bilinen Yeni Test Modeli.

Bölüm 3: Yazılımı Test Etmenin Yeni Bir Yolu - Bu bölümde, diğer araştırmacıların bulgularına dayanarak normal test sürecinde farklı ML yaklaşımlarının nasıl kullanılabileceğini gösterir.

Bölüm 4: Daha Akıllı Test Araçları - Bu bölümde en son test araçları ve ML'nin bunları nasıl daha iyi hale getirdiği ele alınmaktadır. Test uzmanlarının mevcut en iyi araçları kullanabilmesi için hem ödememiz gereken araçları hem de ücretsiz olanları kapsayacaktır.

Bu tez iki ana bölümden oluşmaktadır: teorik ve analitik bölüm ve uygulamalı bölüm. Teorik ve analitik bölümde, AI tekniklerinin yazılım test uygulamalarını iyileştirmedeki potansiyelini inceledik, AI ve ML'nin temel kavramlarına kapsamlı bir genel bakış sağladık ve AI destekli yazılımları test etmekle ilişkili zorlukları ele aldık. Ek olarak, farklı ML yaklaşımlarının yazılım test sürecine nasıl entegre edilebileceğini araştırdık.

Uygulamalı bölümde, Python kodunun kalitesini otomatik olarak sınıflandırmak ve iyileştirmek için makine öğrenimini kullanan bir araç geliştirdik. Bu araç, veri toplamayı, hataları tanımak için bir model eğitmeyi ve geliştiricilere geri bildirim sağlamayı içerir. Flake8 ve Pylint gibi araçları makine öğrenimiyle birleştirerek, bu çalışma kodlama hatalarını otomatik olarak belirlemek ve sınıflandırmak ve geliştiricilere anında geri bildirim sağlamak için yöntemler geliştirdi. Bu projenin önemi, otomatik kod kalite kontrollerine olan artan ihtiyacı karşılamak, kod incelemelerini daha hızlı ve daha doğru hale getirmek, daha iyi yazılımlara ve daha verimli geliştirmeye yol açmaktır.

Bu araştırma, yapay zekanın yazılım testinde nasıl kullanılabileceğini göstererek, potansiyel uygulamaları ve çerçeveleri keşfetmek için AI, ML ve yazılım testi üzerine mevcut araştırmaları eleştirel bir şekilde inceleyen bir literatür tarama metodolojisi kullandı. Bulgular, yazılım test uzmanlarını gelişmiş kod kalitesi, daha hızlı test döngüleri ve sonuçta daha güvenilir yazılım için yapay zekadan yararlanmak için gerekli bilgi ve araçlarla güçlendirebilir.

Anahtar Kelimeler: Analitik Çalışma, Yapay Zeka, Kod Geliştirme Testi, Makine Öğrenimi, Yazılım Testi.

ABSTRACT

Enhancing Code Testing through Artificial Intelligence Techniques: An Applied And Analytical Study

Software testing plays a crucial role in ensuring software quality and reliability. However, traditional testing methods struggle to keep pace with software systems' growing complexity and evolving nature. Artificial intelligence (AI) and machine learning (ML) offer promising avenues to enhance code testing efficiency and effectiveness.

This analytical study investigates the potential of AI techniques in improving software testing practices. It provides a comprehensive overview of the fundamental concepts of AI and ML, along with the specific challenges associated with testing AI-powered software. This research explores how AI can improve the way of code testing. Furthermore, it examines how various ML approaches can be integrated into the software testing process, drawing insights from established research in the field. It also analyzes the impact of ML on modern testing tools, evaluating both commercial and open-source options.

Here's what the research found:

Part 1: The Basics - This part explains what AI and ML are, and the special challenges of testing software that uses AI.

Part 2: Making Sense of AI Frameworks - This part represents two different ways of testing software with AI: the AI for Software Testing Association's model and Paul Gerrard's well-known New Model for Testing.

Part 3: A New Way to Test Software - This part shows how different ML approaches can be used in the normal testing process, based on what other researchers have found.

Part 4: Smarter Testing Tools - This part looks at the latest testing tools and how ML is making them even better. It will cover both tools we have to pay for and free ones, so testers can use the best tools available.

This thesis consists of two main parts: a theoretical and analytical part, and an applied part. In the theoretical and analytical part, we studied the potential of AI techniques in improving software testing practices, providing a comprehensive overview of the fundamental concepts of AI and ML, along with the challenges associated with testing AI-powered software. Additionally, we explored how different ML approaches can be integrated into the software testing process.

In the applied part, we developed a tool using machine learning to automatically classify and improve the quality of Python code. This tool involves collecting data, training a model to recognize errors, and providing feedback to developers. Using tools like Flake8 and Pylint, combined with machine learning, this study developed methods to automatically identify and classify coding errors, providing immediate feedback to developers. The importance of this project lies in meeting the growing need for automated code quality checks, making code reviews faster and more accurate, leading to better software and more efficient development.

By showing how AI can be used in software testing, this research employed a literature review methodology, critically examining existing research on AI, ML, and software testing to explore the potential applications and frameworks. The findings can empower software testers with the knowledge and tools necessary to leverage AI for enhanced code quality, faster testing cycles, and ultimately, more reliable software.

Keywords: Analytical Study, Artificial Intelligence, Enhancing Code Testing, Machine Learning, Software Testing.

TEŞEKKÜR

Yüksek lisans tez yolculuğum boyunca gösterdiğiniz rehberlik ve desteğiniz için yürekten şükranlarımı sunmak için yazıyorum.

Dr. Hossam Yassin, mentorluğunuz düşüncelerimi geliştirmede ve çalışmamın kalitesini arttırmada çok önemli bir rol oynadı. Anlayışlı geri bildirimleriniz beni başarıya yönlendirmede etkili oldu ve bunun için gerçekten minnettarım.

Profesör Dr. Burhan Pektaş, akademik gelişimime yaptığınız katkılardan dolayı en içten teşekkürlerimi sunuyorum. Bilgeliliğiniz ve rehberliğiniz bana mükemmellik için çabalama konusunda ilham verdi ve sizden öğrenme fırsatı bulduğum için minnettarım.

Aileme, özellikle de babama ve anneme olan sarsılmaz sevginiz ve desteğiniz, bu yolculuk boyunca benim dayanağım oldu. Akademik hedeflerimi takip etmemi sağlayan fedakarlıklarınız ve teşvikleriniz için derinden minnettarım.

Bir kez daha ikinize de özveriniz ve desteğiniz için teşekkür ederim. Sizin rehberliğiniz altında çalışma ayrıcalığına sahip olduğum için gerçekten şanslıyım. En içten teşekkürlerimle.

BEYAN FORMU

Bu alıřmadaki bütn bilgi ve belgeleri akademik kurallar erevesinde elde ettiđimi, grsel, iřitsel ve yazılı tm bilgi ve sonuları bilimsel ahlak kurallarına uygun olarak sunduđumu, kullandıđım verilerde herhangi bir tahrifat yapmadıđımı, yararlandıđım kaynaklara bilimsel normlara uygun olarak atıfta bulunduđumu, tezimin kaynak gsterilen durumlar dıřında zgn olduđunu, tarafımdan retildiđini ve řkdar niversitesi Fen Bilimleri Enstits Tez Yazım Kılavuzuna gre yazıldıđını beyan ederim.

5/14/2024

Ahmed RAMADAN

İÇİNDEKİLER

ÖZET	i
ABSTRACT.....	iii
TEŞEKKÜR	v
BEYAN FORMU	vi
İÇİNDEKİLER	vii
TABLolar DİZİNİ.....	xi
ŞEKİLLER DİZİNİ.....	xii
SİMGELER VE KISALTMALAR DİZİNİ	xiii
1. GİRİŞ	1
1.1. Arka Plan	1
1.2. Hedefler	2
1.3. Çalışmanın Gerekçesi	2
1.4. Araştırma Süreci ve Yöntemleri	4
2. GENEL BİLGİLER.....	5
2.1. Mevcut Bilgi ve Mevcut Durum Analizi	5
2.1.1. Yazılım Testinde Durum ve Eğilimler.....	5
2.1.1.1. Anket ve Sonuçlarına Yönelik Eleştiriler	6
2.1.2. Günümüzde Yapay Zeka Odaklı Testin Durumu ve Trendleri.....	7
2.1.3. Yapay Zeka Odaklı Yazılım Testini Destekleyecek Dernekler	7
3. GEREÇ VE YÖNTEM.....	8
3.1. Tezde Kullanılan Araç ve Yöntemler	8
3.2. Metodoloji.....	8
3.3. Yapay Zekayı Test Etme ve Yapay Zeka ile Test Etme - Teorik Arka Plan.....	10
3.3.1. Yapay Zekanın Özellikleri.....	11

3.3.2. Yapay Zeka Testinin Temelleri	14
3.3.3. Agile Gelişim için Yapay Zeka Testinin Faydaları	15
3.3.4. Yapay Zekanın Test Edilmesi - Özellikler ve Zorluklar.....	16
3.3.5. Yapay Zeka Testinin Geleneksel Testlere Etkisi	19
3.3.5.1. Test Yaklaşımı	19
3.3.5.2. Test Teknikleri.....	21
3.3.5.3. Yazılım Test Otomasyonu	22
3.3.5.4. Regresyon Testi	24
3.4. “Yazılım Testi için Yeni Model” – Yapay Zeka Testini Oluşturma Modeli.....	25
3.5. AI Testi – AI Test Sistemi	26
3.5.1. AI Agentlar	27
3.5.2. Keşfetmek ve Algılamak	30
3.5.3. Modelleme	32
3.5.4. Etkileşim	33
3.5.5. Öğrenme.....	35
3.5.5.1. Öğrenmeye Dayalı Teknik.....	35
3.5.6. Yapay Zeka Testinin Riskleri	38
3.6. Test Sürecinde Makine Öğrenmesi Yaklaşımları	40
3.6.1. Planlama, İzleme ve Kontrol	41
3.6.1.1. Üretim Kullanımından Öğrenmek	42
3.6.1.2. Test Senaryosu Önceliklendirmesi	42
3.6.1.3. Tahmine Dayalı Mutasyon Testi.....	43
3.6.1.4. Kaliteli Modeller.....	43
3.6.2. Analiz ve Tasarım.....	44
3.6.2.1. Test Akışı Tasarımı.....	44
3.6.3. Uygulama ve Yürütme.....	45
3.6.3.1. Hizmet Sanallaştırma	46

3.6.3.2. Test Verisi Oluřturma	47
3.6.3.3. Diferansiyel Test.....	48
3.6.3.4. Grsel Test.....	48
3.6.3.5. Testin Uygulanması ve Yrtlmesi iin Bayes Yaklařımları	48
3.6.3.6. Test Oracle	49
4. BULGULAR.....	51
4.1. Yazılım Testi iin Yapay Zeka Araları	51
4.1.1. Test ve Test Yazılımının Ynetimini Destekleyen Aralar	52
4.1.2. Statik Testi Destekleyen Aralar	53
4.1.3. Test Tasarımını, Uygulamayı, Yrtmeyi ve Gnlge Kaydetmeyi Destekleyen Aralar	53
4.1.3.1. Ticari Aralar	54
4.1.3.2. Aık Kaynak Araları	57
4.1.4. Performans lmn ve Dinamik Analizi Destekleyen Aralar.....	59
4.1.5. Diğerk zel Test İhtiyalarını Destekleyen Aralar.....	60
4.1.5.1. Gvenlik Testi.....	62
4.1.6. Genel Amalı Yapay Zeka Araları ve ereveleri	63
4.2. Vaka alıřması: Python Kod Kalitesinin Otomatik Sınıflandırılması ve İyileřtirilmesi iin Makine ğrenimi.....	65
4.2.1. Uygulama Fikri	65
4.2.2. nem.....	65
4.2.3. Uygulama Etkisi	65
4.2.4. Geleneksel Araları Yapay Zeka ile Birleřtirmenin Faydaları	65
4.2.5. Kod Verilerini Toplama, Analiz Etme ve Temizleme	66
4.2.6. Veri Hazırlama ve Model Eğitim İř Akıřı	80
4.2.7. Test İin Kodu Ykleme ve Hazırlama	89
4.2.8. Veri Ynetimi Eğitim ve Yeniden Eğitim İř Akıřı.....	93
4.2.9. Sonu zeti: Veri Toplama Model Eğitim Test Etme ve Yeniden Eğitim	99

4.2.10. Aracın Özellikleri	106
4.2.11. Kısıtlamalar ve Sınırlamalar	107
4.2.12. Sonuç ve Uygulamanın Sonuçları.....	107
5. TARTIŞMA	108
6. SONUÇ ve ÖNERİLER	110
KAYNAKLAR	113
EKLER	120
EK 1. Veri Toplama, Analiz ve Hata Sınıflandırması için Kod	120
EK 2. Kod Sınıflandırması için Makine Öğrenme Modelini Eğitim ve Değerlendirme Kodu.....	126
EK 3. Python Kod Bloklarını Yükleme, Test Etme ve Sınıflandırma Kodu	128
EK 4. Sınıflandırmaları Güncelleme ve Modeli Yeniden Eğitim Kodu	129

TABLÖLAR DİZİNİ

Tablo 1: Test Teknikleri ve AI Geliştirmeleri	22
Tablo 2: Öğrenmeye Dayalı Teknikler ve Faydaları	37
Tablo 3: Yazılım Testine Yönelik Yapay Zeka Araçları	51



ŞEKİLLER DİZİNİ

Şekil 1: Ön Çalışma Planı/Yapısı	9
Şekil 2: Applications of machine learning for software engineering	12
Şekil 3: New Model for Testing	26
Şekil 4: Functional environment of AI test agents	28
Şekil 5: New Model for Software Testing - Exploration Process	32
Şekil 6: New Model for Software Testing Testing Process	34
Şekil 7: Learning based test technique	36
Şekil 8: Test süreci	40
Şekil 9: Oluşturulan Karmaşık Kod Örnekleri ve Veri Toplama ve Temizleme Tamamlandı	9
Şekil 10: Toplanan Veri Örneği	12
Şekil 11: İlk Model Eğitim Doğruluğu.....	26
Şekil 12: İlk Model için Karmaşıklık Matrisi	28
Şekil 13: Hatalı ve Doğru Kodlu Kod Örneği.....	32
Şekil 14: Test Kodu için Sınıflandırma Sonuçları.....	34
Şekil 15: Çeşitli Hata Türleri için Etiketli Veriler	36
Şekil 16: Veri İşleme için Kod Örneği	40
Şekil 17: Yeniden Eğitilmiş Model Doğruluğu.....	36
Şekil 18: Yeniden Eğitilmiş Model için Karışıklık Matrisi	40

SİMGELER VE KISALTMALAR DİZİNİ

Agile: Bu, yazılım geliştirmeye yönelik esnekliği, işbirliğini ve müşteri geri bildirimini vurgulayan yinelemeli bir yaklaşımdır. Agile yöntemler sürekli teslimatı ve iyileştirmeyi teşvik ederek ekipleri değişen gereksinimlere dinamik olarak yanıt vermeye teşvik eder. Agile'in temel ilkeleri arasında çalışan yazılımın sık sık teslim edilmesi, değişen gereksinimlerin benimsenmesi ve sabit bir çalışma temposunun sürdürülmesi yer alır.

AI (Artificial Intelligence): Resmi bir tanım yoktur, ancak yapay zeka sistemleri bağımsız hareket etme ve yeni durumlara uyum sağlama (özerklik ve adaptasyon) yetenekleriyle bilinir.

AI Testi: Bu, yazılım test görevleri için yapay zekanın kullanılması anlamına gelir.

DevOps: Bu yaklaşım, kod yazmadan dağıtıma kadar yazılım geliştirme süreci boyunca işbirliğini ve otomasyonu vurgular.

GPU (Graphical Processing Unit): Bu, makine öğrenimi modellerinin eğitimi de dahil olmak üzere grafiklerin ötesinde görevler için kullanılabilen bir işlemci türüdür.

ISTQB (International Software Testing Qualifications Board): Bu kuruluş, yazılım testi profesyonellerine yönelik sertifikalar sağlar.

MBT (Model Based Testing): Bu test yaklaşımı, bir yazılım uygulamasının veya sisteminin nasıl davranması gerektiğini açıklayan modellere dayanır.

ML (Machine Learning): Bu, sistemlerin belirli görevleri gerçekleştirmek için verilerden öğrendiği bir yapay zeka alanıdır.

NLP (Natural Language Processing): Bu, insan dilini işlemek ve anlamakla ilgilenen makine öğreniminin bir alt alanıdır.

PTP (Predictive Test Prioritization): Bu yöntem, yazılım testlerinin olası etkinliği veya önemine ilişkin tahminlere dayalı olarak önceliklendirilmesini içerir.

QA (Quality Assurance): Bir ürün veya hizmetin kalite standartlarını karşılamasını sağlayan faaliyetleri ifade eder.

TA (Test Automation): Bu, geleneksel olarak testleri çalıştırmak ve sonuçları kontrol etmek için kullanılan yazılım test sürecinin bir kısmını veya tamamını otomatikleştirmek için yazılımın kullanılmasını içerir.

UI (User Interface): Kullanıcı ile dijital ürün veya hizmet arasındaki etkileşim noktasını ifade eder.



1. GİRİŞ

Yapay zeka, bilgisayar biliminde dönüştürücü bir güç haline geldi (Géron, 2019). Ancak yazılım testi üzerindeki etkisi hala gelişmektedir. Test otomasyonu (TA), özellikle Agile ortamlarda yadsınamaz faydalar sunar (Lisa Crispin ve Janet Gregory, 2009), ancak geliştirilmesi ve bakımı son derece zaman alıcı olabilir.

Bu araştırma, yapay zeka tekniklerinin yazılım test sürecini nasıl dönüştürebileceğini araştırıyor ve ML araçlarının TA iş akışlarındaki daha geniş bir görev yelpazesini nasıl otomatikleştirebileceğini araştırıyor. Jeannette Wing gibi sektör liderleri, yapay zekanın veri odaklı yaklaşımının, yazılımları test etme şeklimizi temelden değiştireceğini öngörüyor. Bu araştırma, bu vizyonu gerçeğe dönüştürecek modeller, yöntemler ve araçlar sağlamayı amaçlamaktadır.

1.1 Arka Plan

Yazılım testi, "bir programın sonsuz sayıda olasılığa karşı sonlu sayıda testle doğrulanması" sürecidir (Myers et al., 2011). Yazılımın karmaşıklığı arttıkça manuel testlerin etkinliği giderek azalır. Yapay zeka ve akıllı test otomasyonu, daha kapsamlı test kapsamı elde etmek için bir çözüm olarak görülüyor.

Otomasyonun artırılmasına açık bir ihtiyaç var, ancak yapay zekanın TA'ya dahil edilmesiyle ilişkili potansiyel riskleri de kabul etmeliyiz. Bu araştırma bu potansiyel riskleri ele alacak ve azaltma stratejileri önerecektir.

Anahtar sorular:

- Yazılım testi için ne tür AI etkili bir şekilde kullanılabilir?
- AI, çeşitli düzeylerde testlerin planlanmasına, yürütülmesine ve yazılım kalitesinin değerlendirilmesine nasıl yardımcı olabilir?
- AI, test sonuçlarını doğrulamak için gerekli güvenilirlik düzeyine ulaşabilir mi?
- AI ile otomasyona en uygun test görevleri hangileridir?

Bu araştırma, yapay zekanın entegre edilmesinin yazılım geliştirme süreçlerini, test iş akışlarını ve geliştirme ekiplerinin kendisini nasıl etkileyebileceğini inceleyecek. DevOps ve Agile metodolojileri test otomasyonu uygulamalarını büyük ölçüde etkilediğinden, bulgular bu alanlardaki profesyoneller için değerli olacaktır.

1.2 Hedefler

- Yazılım testi iş akışlarında yapay zekadan yararlanmak için pratik öneriler geliştirin.
- Test uzmanları ve geliştiriciler arasında ortak bir dil geliştirerek bu yöntemleri yerleşik test çerçevelerine entegre edin.
- ISTQB terminolojisine bağlı kalarak AI araçlarını ve tekniklerini belirli test alanlarıyla uyumlu hale getirin (International Software Testing Qualifications Board, 2023) .

Bu araştırmanın amacı sadece öneri sunmanın ötesine geçmektir. Amaç, yapay zekayı standartlaştırılmış test uygulamalarının ayrılmaz bir parçası haline getirmek, birleşik bir dili teşvik etmek ve AI ilerlemeleri ile gerçek dünyadaki test uygulamaları arasındaki boşluğu kapatmaktır. Bu çalışma, AI araçları ile yerleşik test konseptleri arasında bağlantılar kurarak onu yazılım geliştirme ekipleri için değerli bir rehber haline getirecek. Mevcut araştırmaları analiz ederek ve somut stratejiler önererek amaç, ekipleri yapay zekanın potansiyelinden yararlanma ve verimli, yüksek kaliteli yazılım testinde yeni bir çağ başlatma konusunda güçlendirmektir.

1.3 Çalışmanın Gerekçesi

Yapay zekanın yazılım testlerinde artan kullanımı hem inanılmaz olasılıkları hem de zorlukları beraberinde getiriyor. Yapay zekadan gerçek anlamda yararlanmak için test topluluğunun sürekli araştırma yapması ve bilgiyi paylaşması gerekir. Çalışma şu nedenlerden dolayı önemlidir:

- **Hızlı Gelişmeler:** Dünya Kalite Raporu, yapay zekanın kalite güvencesinde artan önemini vurguluyor ve bu alanın ne kadar hızlı geliştiğini gösteriyor. Yapay zekanın tüm potansiyelinden yararlanmak için bu gelişmelere ayak uydurmak çok önemlidir (World Quality Report, 2022).
- **Farkındalık Açığı:** Pek çok kuruluş test için yapay zeka kullanıyor (LambdaTest'in 2023 anketine göre %78), ancak çok daha azının (yalnızca %45) net bir planı var (LambdaTest, 2023). Bu, bazılarının AI'nın yeteneklerini veya onu etkili bir şekilde nasıl uygulayacaklarını tam olarak anlamadan benimseyebileceğini gösteriyor. Ayrıca InfoQ'nun 2023 raporu ("Yazılım Testi, Yapay Zeka ve Makine Öğrenimi Trendleri") birçok test uzmanının yapay zeka test araçlarını kullanma becerisine sahip olmadığına işaret ediyor (InfoQ, 2023). Bu, farkındalık ile etki arasındaki boşluğu kapatmak ve sonuçta testlerde yapay zeka kullanımını optimize etmek için eğitim ve bilgi paylaşımı ihtiyacını vurguluyor.
- **Bilgi Anahtardır:** Yapay zekanın testlerde başarılı bir şekilde uygulanması, onun güçlü, zayıf yönlerinin ve en iyi uygulamalarının anlaşılmasını gerektirir. Bu çalışma, son araştırmalardan, başarılı stratejilerden ve kaçınılması gereken potansiyel engellerden elde edilen bilgiler sağlayarak bu bilgi boşluğunu doldurmayı amaçlamaktadır.

Sonuç olarak, AI testinin hızlı gelişimi, farkındalık ile etki arasındaki uçurum ve pratik rehberliğe duyulan ihtiyaç, sürekli araştırma ve bilgi paylaşımının önemine işaret ediyor. Bu çalışma, araçlar ve teorik yaklaşımlar arasındaki boşluğu doldurmayı, sonuçta kuruluşların yapay zekayı test süreçlerine etkin bir şekilde entegre etmelerine ve yazılım kalitesi açısından dönüştürücü potansiyelini açığa çıkarmalarına yardımcı olmayı amaçlıyor.

1.4 Araştırma Süreci ve Yöntemleri

Bu araştırmaya başladığımda yapay zeka ve yazılım testinin birlikte nasıl çalıştığını kapsamlı bir şekilde araştıran çok fazla kaynak yoktu. Bu boşluğu gidermek için test ve yapay zeka süreçlerini, tekniklerini, araçlarını ve potansiyel kullanım örneklerini içeren modern bir anahtar kelime yapısı oluşturdum. Bu çerçeve, ISTQB sertifikasyon programındaki terminoloji ve standartlar kullanılarak oluşturulmuştur (ISTQB, 2018).

ISTQB güçlü bir temel sağlarken, yapay zeka testiyle ilgili tartışmalara dahil olmak da çok önemliydi. Mevcut terimlerin yeterli olmadığı durumlarda, güncel kaynaklardan terimler ve kısaltmalar ödünç alarak bunları dikkatli bir şekilde en son ISTQB standartları bağlamına entegre ettim.

Bazıları geleneksel test uygulamalarının Agile geliştirme modellerine uyum sağlayıp sağlayamayacağını sorguladı. Ancak ISO Yazılım Testi Çalışma Grubuna (WG26) liderlik eden Dr. Stuart Reid'in son görüşleri bu kaygıları çürütmektedir (Reid, 2014). Dr. Reid, standartların Agile yaklaşımlarla iyi çalışabileceğini savunuyor ve dokümantasyon uygulamalarında izin verilen esnekliğin altını çiziyor. Bu modern bakış açısı, yapay zekanın gerçek dünyadaki Agile test ortamlarında nasıl uygulanabileceğini araştıran bu araştırmayla mükemmel bir şekilde uyum sağlıyor.

Bu araştırmada kullanılan ayrıntılı metodoloji ve spesifik araçlar 3. bölümde daha ayrıntılı olarak ele alınacaktır.

2. GENEL BİLGİLER

Bu bölümde, mevcut bilgilere işaret ederek ve AI-odaklı yazılım testinin durumunu analiz ederek çalışma konusu hakkında kapsamlı bir arka plan sunacağız. Yazılım test süreçlerini iyileştirmek için yapay zeka kullanmanın önemini vurgulayarak, bu alandaki güncel trendleri ve son gelişmeleri gözden geçireceğiz.

2.1. Mevcut Bilgi ve Mevcut Durum Analizi

AI-odaklı testlerin mevcut durumu incelendi ve çeşitli kaynaklardan bilgiler elde edildi. Sektör trendlerini, kilit oyuncularını ve yapay zekanın testlerde gelişen rolünü inceliyor. Bu mevcut bilgiyi gözden geçirerek ve mevcut durumu analiz ederek, bu alandaki mevcut eğilimleri, kilit sektör oyuncularını ve yapay zekanın test süreçlerindeki rolünün nasıl geliştiğini vurgulayacağız.

2.1.1. Yazılım Testinde Durum ve Eğilimler

Dünya Kalite Raporu değerli bir referans olmaya devam ederken, son araştırmalar AI testinde heyecan verici eğilimleri ortaya koyuyor (World Quality Report, 2022). Sektörün odak noktası yalnızca verimliliğin ötesine kayıyor ve AI'nın daha geniş uygulamalarını kapsıyor:

- **Güvenlik Testi:** AI, güvenlik açıklarını tespit etmek, tehditleri proaktif olarak azaltmak ve siber saldırılara karşı daha dayanıklı yazılımlar oluşturmak için kullanılıyor (OWASP, 2020).
- **Açıklanabilir AI:** AI kararlarının şeffaflığı ve anlaşılması, güven oluşturmak ve önyargıyı azaltmak için çok önemlidir (Harvard Business Review, 2022).
- **Performans ve Ölçeklenebilirlik Testi:** AI, yük altında sistem davranışını analiz eder, darboğazları tahmin eder ve bulut dağıtımları için ölçeklenebilirliği optimize eder (The Linux Foundation, 2021).

Bu gelişmeler, AI'nın rolünün verimlilik kazanımlarının ötesine geçerek güvenliği, açıklanabilirliği ve performans optimizasyonunu nasıl kapsadığını vurgulamaktadır. Ek olarak, AI'nın Robotik Süreç Otomasyonu (RPA) ve diğer otomasyon teknolojileriyle bir araya gelmesi, kendi kendini iyileştiren sistemler ve düşük kodlu/kodsuz araçlarla "hiperotomatik" test ortamlarını teşvik ediyor (Forrester, 2023; George, A et al., 2023).

AI modellerinin sürekli öğrenme ve geri bildirim döngülerini kullanma yeteneği, bağlam farkında test ve gerçek zamanlı geri bildirim gibi özelliklerin giderek daha önemli hale gelmesiyle daha fazla yeniliğe yol açıyor (MIT Technology Review, 2023; Microsoft Azure, 2023).

2.1.1.1. Anket ve Sonuçlarına Yönelik Eleştiriler

Dünya Kalite Raporu anketi değerli içgörüler sunar (World Quality Report, 2022), ancak testte AI benimsenmesinin daha doğru bir resmini elde etmek için sınırlamalarını göz önünde bulundurmak önemlidir:

- **Sınırlı Kapsam:**
 - **Endüstri Önyargısı:** Büyük şirketlere odaklanmak, daha küçük şirketlerin ve yeni kurulan şirketlerin bakış açılarını kaçırabilir.
 - **Coğrafi Önyargı:** Katılımcıların küresel bir yayılımı olmadan, AI test benimsenmesindeki bölgesel farklılıklar göz ardı edilebilir.
- **Katılımcı Odaklı:**
 - **Yönetim Bakış Açısı:** Yöneticilerin ve müdürlerin baskınlığı, sahadaki bireysel testçilerin deneyimlerini yakalayamayabilir (InfoQ, 2023).

Bu sınırlamalara rağmen, anketin AI'nın verimlilik potansiyeli ve veri analizi becerilerine duyulan ihtiyaç hakkındaki temel bulguları geçerliliğini korumaktadır. Gelecekteki çalışmalar, AI testinin mevcut durumu hakkında daha eksiksiz bir resim sağlamak için daha geniş bir temsil hedeflemelidir.

2.1.2. Günümüzde Yapay Zeka Odaklı Testin Durumu ve Trendleri

AI testlerinin gelişen ortamı, başarılı entegrasyon için sürekli adaptasyon ve beceri geliştirmeyi gerektiriyor:

- **Beceri Dönüşümü:** Test uzmanlarının beceri setlerini veri analizi, AI'nın temel bilgisi ve belirli AI test araçlarında yeterlilik kapsayacak şekilde genişletmeleri gerekir.
- **İşbirliğinin Gücü:** En etkili yaklaşım, geleneksel test uzmanlarının uzmanlıklarının ve AI yeteneklerinin birleşik güçlü yanlarından yararlanarak sinerjik bir test modeli oluşturur.
- **Veri Sorumluluğu:** Veri yönetimi uygulamalarını anlamak, gizlilik düzenlemelerine uymak ve olası önyargılara karşı dikkatli olmak, etik ve sorumlu AI testi için hayati önem taşır.

2.1.3. Yapay Zeka Odaklı Yazılım Testini Destekleyecek Dernekler

Gelişen AI test topluluğu, çeşitli derneklerin ve girişimlerin katkılarından faydalanır:

- **Açık Kaynak Şampiyonları:** Kuruluşlar ve bireyler, açık kaynaklı test araçlarına ve çerçevelerine aktif olarak katkıda bulunarak inovasyonu ve yaygın benimsenmeyi hızlandırır.
- **Akademik Güç Merkezi:** Üniversiteler, araştırma, deney ve yayınlar aracılığıyla bilgi paylaşımı yoluyla AI testinin sınırlarını zorlamada kritik bir rol oynar.

3. GEREÇ VE YÖNTEM

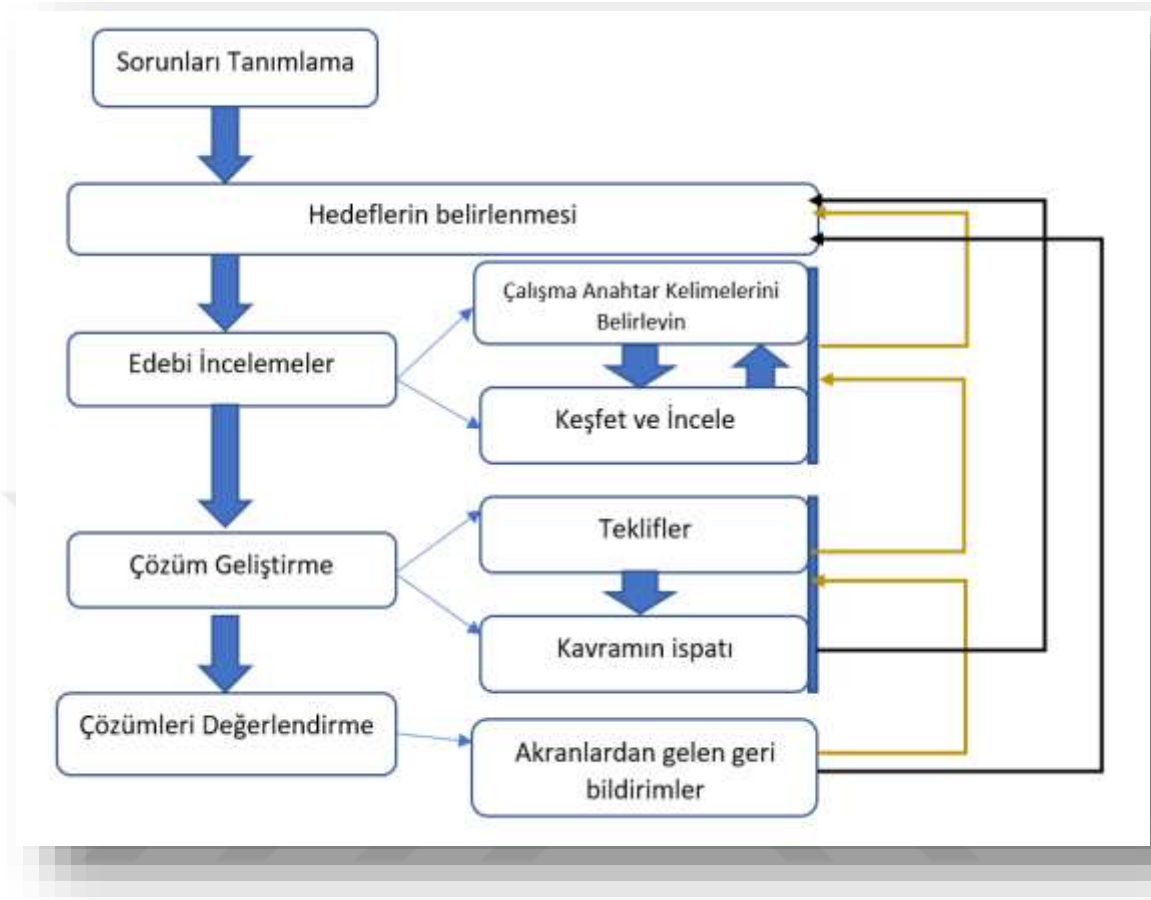
Bu bölümde bu çalışmada kullanılan araçlar ve metodolojiler açıklanmaktadır. AI testinin kapsamını keşfetmek için araçları ve teorik yaklaşımları birleştirir.

3.1. Tezde Kullanılan Araç ve Yöntemler

Araştırma hedeflerine ulaşmak için çeşitli araç ve tekniklerden yararlandı. Bunlar şunları içerir:

- **AI Test Araçları ve Çerçeveleri:**
 - **Ticari Araçlar:** Yazılım testinin çeşitli yönleri için kullanılan özel AI araçları.
 - **Açık Kaynak Araçları:** AI-odaklı testleri destekleyen ücretsiz araçlar ve çerçeveler.
- **Makine Öğrenimi Modelleri:**
 - **Sınıflandırma Tabanlı Sanallaştırma (CBV):** Eğitilmesi daha hızlı olan RIPPER algoritmasını kullanır.
 - **Sıradan Sıraya Dayalı Sanallaştırma (SSBV):** Daha uzun eğitim süreleri ve GPU'lar gibi güçlü donanım gereksinimlerine rağmen, daha yüksek doğruluk için Uzun Kısa Süreli Bellek (LSTM) ağlarını kullanır.
- **AI Agentlar:**
 - **Koordinatör Agentsi:** Tüm test sürecini yönetir, kopyaları ortadan kaldırır ve görevleri atar.
 - **Test Agentsi:** Test senaryolarını keşfetmek, oluşturmak ve çalıştırmak için yazılımla etkileşime girer.
- **AI Test Ortamı:**
 - **Hizmet Sanallaştırma:** Eksik veya güvenilir olmayan hizmetleri simüle etmek için kullanılır ve hizmet davranışlarını modellemek ve kopyalamak için makine öğreniminden yararlanır.
 - **Bayesian Modelleri:** Efor tahmini, test planlaması ve güvenilirlik takibi dahil olmak üzere yazılım kalite analizi için uygulanır.

3.2. Metodoloji



Şekil 1 – Ön Çalışma Planı/Yapısı

Bu araştırmanın metodolojisi, Şekil 1 - Ön Çalışma Planı/Yapısında gösterildiği gibi aşamalı bir yaklaşımla yapılandırılmıştır. Bu plan, yapay zekayı yazılım testine entegre etmek için sistematik bir sürecin ana hatlarını çiziyor.

Aşama 1: Sorunların Belirlenmesi ve Hedeflerin Belirlenmesi

- Araştırma hedeflerini ve kapsamını tanımlayın.
- AI ve yazılım testi entegrasyonundaki temel zorlukları belirleyin.

Aşama 2: Literatür Taraması ve Anahtar Kelime Araştırması

- Mevcut bilgiyi toplamak için kapsamlı bir literatür taraması yapın.
- İlgili araştırma ve metodolojilerin araştırılmasına rehberlik edecek bir anahtar kelime yapısı geliştirin.

Aşama 3: Çözüm ve Tekliflerin Geliştirilmesi

- Literatür incelemesine dayanarak, yazılım testinde yapay zekanın kullanılmasına yönelik potansiyel çözümler ve metodolojiler önerin.
- Bu önerileri destekleyecek teorik modeller ve çerçeveler geliştirin.

Aşama 4: Çözümlerin ve Geri Bildirimin Değerlendirilmesi

- Önerilen çözümleri etkinlik ve fizibilite açısından değerlendirin.
- Uzmanlardan geri bildirim toplayın ve modelleri ve metodolojileri buna göre geliştirin.

Bu yapılandırılmış metodolojiyi takip eden çalışma, araçlar ve teorik yaklaşımlar arasındaki boşluğu doldurmayı, sonuçta kuruluşların yapay zekayı test süreçlerine etkili bir şekilde entegre etmelerine ve yazılım kalitesi açısından dönüştürücü potansiyelini ortaya çıkarmalarına yardımcı olmayı amaçlıyor.

3. 3. Yapay Zekayı Test Etme ve Yapay Zeka ile Test Etme - Teorik Arka Plan

Bu bölüm, iki ana alana ayrılabilen AI test dünyasını inceler:

1. **Yazılımı Test Etmek İçin AI Kullanma:** Bu, yazılım programlarındaki hataları ve kusurları bulabilen AI sistemleri geliştirmeyi içeren yeni bir alandır.
2. **AI Sistemlerinin Kendilerini Test Etme:** Bu, AI sistemlerinin beklendiği gibi çalıştığından ve beklenmeyen sonuçlar üretmediğinden emin olmak anlamına gelir.

Zamanla öğrenen ve uyum sağlayan AI sistemleri, geleneksel yazılımlara kıyasla test için farklı bir yaklaşım gerektirir. Bu kavram, yazılımın evrimleşirken kendisini sürekli olarak hatalara karşı kontrol ettiği "kendi kendini test etme"yi içeriyordu. Kendi kendini test etme fikri Tariq'ten geliyor. Yazılıma doğrudan yerleştirilecek kendi kendini yapılandıran, kendi kendini iyileştiren, kendi kendini optimize eden ve kendi kendini koruyan "test botları" oluşturmayı öneriyor (Tariq M, 2018).

Bir sonraki bölümde Yapay Zeka daha ayrıntılı olarak incelenecek ve farklı Makine Öğrenmesi yaklaşımlarına bakılacaktır.

3.3.1. Yapay Zekanın Özellikleri

Yapay zeka kafa karıştırıcı bir konu olabilir ve herkesin üzerinde anlaştığı tek bir tanım yoktur. Yapay zekayı anlamak için daha iyi bir yol, temel özelliklerine bakmak olabilir: kendi başına hareket edebilme (özerklik) ve uyum sağlayabilme (uyarlanabilirlik).

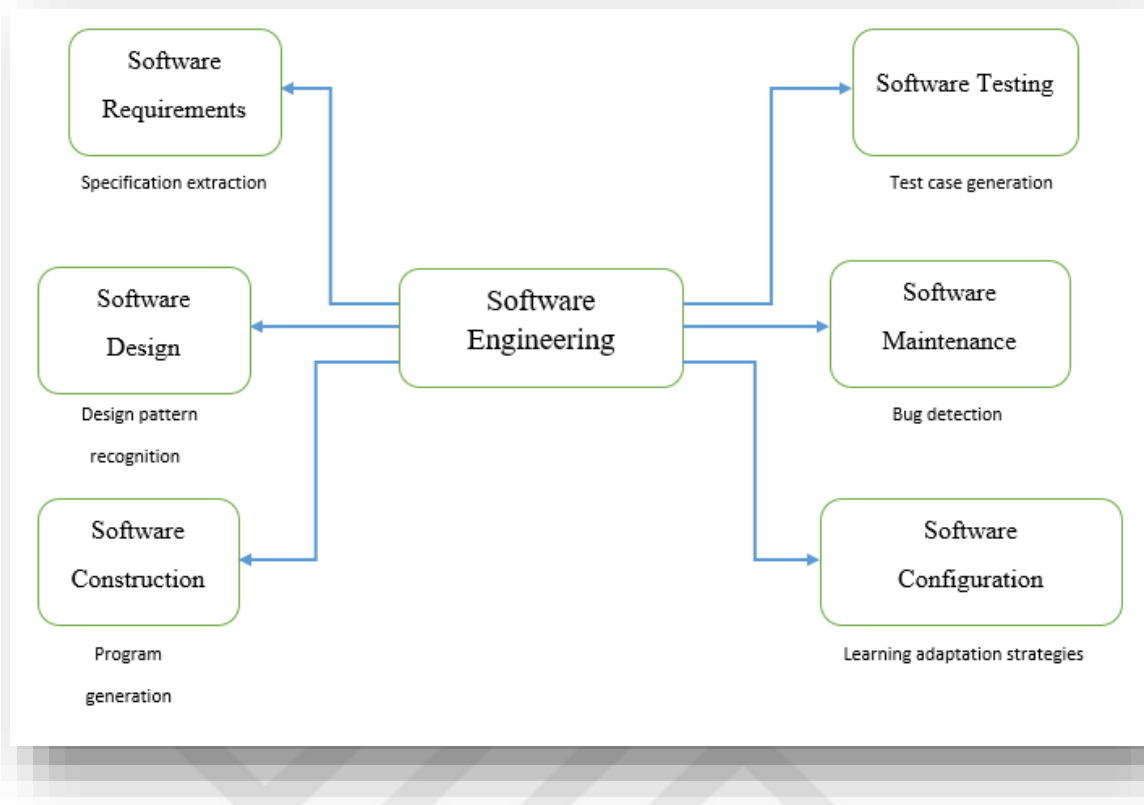
Bilgisayar dünyasında, yapay zeka bilgisayar biliminin bir dalıdır. Makine öğrenimi, derin öğrenme (beynin yapısı ve işlevinden esinlenen bir yazılım türü), veri bilimi ve robotik (tüm bu AI alanlarından fikirler kullanan) gibi diğer alanlarla yakından ilişkilidir. (University of Helsinki & Reaktor, 2024)

Yapay zekayı tanımlamanın bir yolu, akıllı bir agent olarak tanımlamaktır. Bu agent, sensörler aracılığıyla çevresini algılayabilir ve ortamında belirlenen belirli hedeflere ulaşmak için eylemlerde bulunabilir. agentın hedeflerini belirleme şekli, nasıl inşa edildiğine bağlıdır. Örneğin, basit bir agent sadece temel kuralları takip edebilirken, daha gelişmiş bir öğrenme agentı, ortamından öğrendiklerine dayanarak kendi hedeflerini belirleyebilir. (Stuart & Peter, 2016)

Makine öğrenimi bir tür yapay zekadır, ancak bilgisayar biliminin bir parçası olduğunu ve bazı alanların istatistiklerle bile örtüştüğünü söylemek daha doğru olur. Makine öğrenimi, yapay zeka sistemlerinin verilerden modeller oluşturarak uyum sağlamasını sağlar. Bir makine öğrenimi sisteminin öğrenmesi gereken veri ne kadar fazlaysa, performansı o kadar iyi olur. Bu tür öğrenme, yapay zekayı robotikte mümkün kılan şeydir (University of Helsinki & Reaktor, 2024).

Yazılım Mühendisliği ve Makine Öğrenimi

Makine öğreniminin yazılım mühendisliği süreci boyunca nasıl uygulanabileceğini göstermek için. Aşağıdaki diyagram 2017'de bir yazılım mühendisliği konferansında sunulmuştur (Meinke & Bennaceur, 2017). Makine öğreniminin başarıyla kullanıldığı yazılım mühendisliğinin farklı alanlarını göstermektedir. Diyagramda görüldüğü gibi, makine öğrenimi yazılım testinde, test vakaları oluşturmaktan planlamaya, önceliklendirmeye ve testleri çalıştırmaya kadar her şeyde giderek daha fazla kullanılmaktadır.



Şekil 2 – Applications of machine learning for software engineering
(Meinke & Bennaceur, 2017)

- **Gereksinim Spesifikasyonunun Çıkarılması:** Yapay zekanın bir alt alanı olan doğal dil işleme, yazılım gereksinimlerinin yazılı açıklamalarını otomatik olarak bilgisayarlar tarafından anlaşılabilir bir formata dönüştürmek için kullanılabilir. Bu, yazılım mühendislerinin zamandan ve emekten tasarruf etmesini sağlayabilir ve aynı zamanda gereksinimlerin açık ve net olmasını sağlamaya da yardımcı olabilir.
- **Tasarım Modeli Tanıma:** Makine öğrenimi, mevcut yazılım tasarımlarındaki kalıpların belirlenmesine yardımcı olabilir ve yeni projeler için uygun tasarım modelleri önerebilir. Bu, yazılım mühendislerinin daha verimli ve bakımı kolay yazılım geliştirmelerine yardımcı olabilir.
- **Program Oluşturma:** AI, spesifikasyonlara veya örneklere dayalı olarak otomatik olarak kod oluşturabilir. Bu, özellikle tekrarlanan görevler için yazılım mühendisleri için büyük bir zaman tasarrufu sağlayabilir.

- **Test Senaryosu Oluřturma:** Makine öğrenimi, bir yazılım programının farklı bölümlerini test etmek amacıyla otomatik olarak test senaryoları oluşturmak için kullanılabilir. Bu, programın tüm parçalarının kapsamlı bir şekilde test edilmesini sağlayarak yazılımın kalitesini artırmaya yardımcı olabilir.
- **Test Verisi Seçimi:** Yapay zeka, yazılım testi için kullanılacak en etkili test verilerinin seçilmesine yardımcı olabilir. Bu, test için gereken zamanı ve çabayı azaltmaya yardımcı olabilir ve aynı zamanda testin etkinliğini artırmaya da yardımcı olabilir.
- **Kusur Tahmini:** Makine öğrenimi modelleri, bir yazılım programının hangi bölümlerinin hata içirme olasılığının en yüksek olduğunu tahmin edecek şekilde eğitilebilir. Bu, yazılım mühendislerinin dikkatlerini programın hatalı olma olasılığı en yüksek olan kısımlarına odaklamalarına yardımcı olabilir.

Yazılım mühendisleri, makine öğrenimini bu yollarla kullanarak yazılımın verimliliğini ve etkinliğini artırabilir.

Yapay zekada kullanılan iki ana makine öğrenimi türü vardır: denetimli öğrenme ve denetimsiz öğrenme.

- **Denetimli öğrenme** bir öğretmene sahip olmak gibidir. Yapay zeka sistemine halihazırda etiketleri veya sınıflandırmaları olan veri örnekleri veriliyor. Sistem, bu verilerdeki kalıpları tanımayı öğrenir ve ardından bu kalıpları yeni verileri sınıflandırmak için kullanır. Örneğin, bir algoritma bir grup etiketli kedi resmi üzerinde eğitilmiş olabilir ve daha sonra size yeni bir resmin kedi olup olmadığını söyleyebilir.
- **Denetimsiz öğrenme** farklıdır çünkü verilerde herhangi bir etiket yoktur. Bu, sistemin ne kadar iyi çalıştığını bilmeyi zorlaştırır. Denetimsiz öğrenme, benzer şeyleri bir arada gruplamak gibi verilerdeki kalıpları bulmak için kullanışlıdır. Örneğin benzer satın alma alışkanlıklarına sahip müşterileri bir arada gruplamak için kullanılabilir.

Yukarıdaki iki yaklaşımı birleştiren, yarı denetimli öğrenme adı verilen üçüncü bir öğrenme türü de vardır. Yarı denetimli öğrenmede sistem, etiketlenmiş bazı veriler ve etiketlenmemiş bazı veriler üzerinde eğitilir. Bu, çok fazla etiketlenmemiş veri içeren bir sistemi başlatmak için daha az miktarda etiketli veri kullanarak eğitmenin iyi bir yolu olabilir. (Géron, A.,2019)

3.3.2. Yapay Zeka Testinin Temelleri

Çevrimiçi bir mağazaya bakan bir insan düşünün. Beynimiz çoğu mağazada ortak olan düzeni ve özellikleri doğal olarak anlar: resimli ürün listeleri, açıklamalar, arama çubukları, alışveriş sepetleri, oturum açma bilgileri vb. AI testi de benzer şekilde çalışır ve test edilen yazılımın üzerinde ekstra bir anlayış katmanı oluşturur (Lima et al., 2020). Amaç, konu yazılım davranışı olduğunda yapay zekanın insan beyni gibi düşünmesini ve hareket etmesini sağlamaktır.

Uzmanlara göre bu test yaklaşımı çeşitli avantajlar sunuyor. AI testi çok yönlüdür ve birçok farklı amaç için kullanılabilir. Aynı zamanda yeniden kullanılabilir, yani aynı testler birden fazla yazılım programına uygulanabilir. Ek olarak, yapay zeka testleri sağlamdır (çeşitli durumlarda iyi çalışır) ve uyarlanabilirdir, yani yazılımdaki değişikliklere uyum sağlayabilir. Son olarak yapay zeka testi ölçeklenebilir ve büyük ve karmaşık yazılım projelerini yönetebilir (Tariq M, et al., 2019).

AI testlerinde kullanılan güçlü tekniklerden biri makine öğrenimidir. Bu, yapay zekanın farklı uygulama türleri ve endüstrilerde yazılım kalitesinin çeşitli yönlerini test etmesine olanak tanır. Bu testler belirli yazılımlardan bağımsız olduğundan, aynı testler benzer bir alandaki birçok uygulamada kullanılabilir, hatta tamamen farklı alanlarda kullanılmak üzere uyarlanabilir. Örneğin, bir alışveriş sepetine veya giriş formuna ürün ekleme işlevselliğinin test edilmesi birçok farklı uygulamaya uygulanabilir.

Makine öğrenimi aynı zamanda "pestisit paradoksu" olarak adlandırılan bir zorluğun üstesinden gelmenin bir yolunu da sunuyor. Bu, aynı testler tekrar tekrar çalıştırıldığında meydana gelir ve geleneksel test otomasyonunun yeni hataları kaçırmasına neden olur. Bir tür makine öğrenimi olan takviye öğrenimi, yapay zekanın sürekli olarak yeni testler oluşturmaya olanak tanıyarak yazılımın aynı testlere alışmasını ve potansiyel olarak yeni sorunları gizlemesini önler. AI testi oluşturmayı buluttan birçok test programı

çalıştırmayla birleştirerek, testlerin genel kapsamlılığı ve etkinliği önemli ölçüde iyileştirilebilir (Tariq M, et al., 2019).

Özünde AI testi, yazılım geliştirme araç kutusuna güçlü bir araç ekler. Yazılım davranışının insan benzeri anlayışını taklit etmeye yardımcı olur ve yeniden kullanılabilirlik, uyarlanabilirlik ve büyük ölçekli projeleri yönetebilme yeteneği gibi çeşitli avantajlar sunar. AI testi kapsamındaki ML, daha geniş test kapsamına olanak tanır ve geleneksel test yöntemlerindeki sınırlamaların aşılmasına yardımcı olur.

3.3.3. Agile Gelişim için Yapay Zeka Testinin Faydaları

Agile geliştirme, yazılım oluşturma'nın popüler bir yolu haline geldi çünkü esnekliği ve müşteri ihtiyaçlarına yanıt vermeyi ön planda tutuyor.

Evrak işleriyle boğuşan geleneksel yöntemlerin aksine, Agile geliştirme, sürekli geri bildirim ve ayarlamalarla başarılı olur. Agile geliştirmenin yakın bir arkadaşı olan DevOps, yazılım oluşturma (geliştirme) ile yazılımın sorunsuz çalışmasını sağlama (işlemler) arasındaki boşluğu doldurur.

Agile gelişimin kalbinde on iki temel ilke vardır; en üst düzeydeki faturalandırma müşterileri mutlu edecek ve düzenli olarak değerli özellikler sunacaktır (Authors of the Agile Manifesto, 2001). AI testinin roket yakıtını bu temel ilkelere nasıl dahil ettiği aşağıda açıklanmıştır:

- **Sürekli Teslimat ile Müşterileri Mutlu Tutmak:** Siz yeni özellikler geliştirirken yazılımınızda aksaklıklar olup olmadığını sürekli kontrol eden yorulmak bilmez bir yardımcı hayal edin. AI testinin büyüğü budur. AI sistemleri bağımsız olarak çalışabildiğinden, yazılım üzerinde sürekli olarak testler yürütebilir ve yeni özelliklerin ve güncellemelerin müşterilere ulaşmadan önce mükemmel şekilde çalışmasını sağlayabilir. Bu sürekli geri bildirim döngüsü, hataların gözden kaçması ve kullanıcıların sinirlenmesi riskini en aza indirir.
- **Anında Değişen İhtiyaçlara Uyum Sağlamak:** Tıpkı Agile geliştirme gibi, AI testi de tamamen uyum sağlamayla ilgilidir. Bir müşteri yeni bir özellik talep ederse AI test sürecini buna göre ayarlayabilir. Bu, mevcut tüm işlevlerin yeni eklemeyle sorunsuz bir şekilde çalışmasını sağlayarak beklenmedik arızaların

önlenmesini sağlar. Geleneksel test yöntemleri, değişikliklere uyum sağlamak için tüm test takımlarının yeniden yazılmasını gerektirebilir, bu da geliştirme sürecini yavaşlatır. AI testi bu engeli ortadan kaldırarak Agile ekiplerin çevik ve duyarlı kalmasını sağlar.

- **Daha Hızlı, Daha Sık Teslimat:** Yazılımı elle test etmek için harcadığınız onca zamanı hatırlıyor musunuz? AI testi, bu süreci önemli ölçüde otomatikleştirerek geliştiricilerin yeni özellikler oluşturmaya odaklanmasına olanak tanır. Ek olarak yapay zekanın yazılımdaki değişikliklere uyum sağlama yeteneği, testlerin sürdürülmesi için gereken zamanı ve çabayı azaltır. Bu, mutlu müşteriler için daha hızlı geliştirme döngüleri ve daha sık yazılım güncellemeleri anlamına gelir.
- **En Önemli Şeye Odaklanmak:** Agile geliştirme, yüksek kaliteli yazılım sağlamak için kapsamlı testlerin önemini vurgular. Ancak manuel test, zaman alıcı ve tekrarlanan bir görev olabilir. AI testi, bu iş yükünün büyük bir kısmını üstlenerek insan test uzmanlarının uzmanlıklarını yapay zekanın yetersiz kaldığı alanlara odaklamasına olanak tanır. Bu, karmaşık kullanıcı etkileşimlerinin veya insan dokunuşu gerektiren yaratıcı işlevlerin test edilmesini içerebilir.
- **Ekip Olarak Sürekli Gelişmek:** Agile gelişimin temel ilkesi, ekipleri daha verimli çalışma yolları bulmaya ve gereksiz görevleri ortadan kaldırmaya teşvik eder. AI ile test etme gibi tekrarlanan görevleri otomatikleştirmek, değerli insan uzmanlığını daha karmaşık işler için serbest bırakır. Bu, insanların yapay zekanın yapamadıklarına odaklanabilmesi için otomatikleştirilebilecekleri otomatikleştirme ilkesiyle mükemmel bir şekilde uyum sağlıyor. Agile ekipler AI testinden yararlanarak iş akışlarını kolaylaştırabilir, iyileştirilecek alanları belirleyebilir ve daha iyi yazılım geliştirme uygulamaları için sürekli çaba gösterebilir.

3.3.4. Yapay Zekanın Test Edilmesi - Özellikler ve Zorluklar

AI sistemlerinin amaçlandığı gibi çalıştığından emin olmak, sıradan yazılım testlerinden daha fazlasını gerektirir. AI sistemleri genellikle karmaşıktır, sürekli uyum sağlar ve çıktıları tahmin edilemez olabilir (Tariq M, et al., 2019). Yapay zekayı test ederken karşılaştığımız engellerin bir dökümü aşağıda verilmiştir:

- **Zor Sonuçlar ve Veri İkilemleri:** Müşteri duyarlılığını anlamak için yorumları analiz eden bir program hayal edin. Aynı inceleme bir AI tarafından olumlu, diğeri tarafından olumsuz görülebilir. Bu tutarsızlık yapay zeka sistemlerinin nasıl eğitildiğinden kaynaklanmaktadır. Eğitim verilerinin kalitesi ve miktarı büyük bir rol oynamaktadır. Eğitim verileri taraflı veya eksikse yapay zeka yanlış veya adil olmayan kalıpları öğrenebilir. Bu sorunu çözmek için test uzmanlarının yapay zekayı geliştirmek için kullanılan eğitim verilerini dikkatlice incelemesi ve çeşitli ve yüksek kalitede olmasını sağlaması gerekir. Diğer bir zorluk ise birçok yapay zeka sisteminin veriye aç olmasıdır. Etkili bir şekilde öğrenmek ve çalışmak için büyük miktarda veriye ihtiyaç duyarlar. Bu veriler genellikle kalıpları tanımlayan ve tahminlerde bulunan karmaşık algoritmaları eğitmek için kullanılır. Buradaki sorun, eğer eğitim için kullanılan veriler önyargılıysa, yapay zeka bu önyargıları devralabilir ve adil olmayan veya ayrımcı kararlar alabilir. Örneğin, önyargılı kredi başvuru verileri üzerine eğitilmiş bir yapay zeka, belirli demografik grupların kredi başvurularını sistematik olarak reddedebilir. Bu tür sorunları önlemek için test uzmanlarının yapay zekayı oluşturmak için kullanılan eğitim verilerinin yalnızca miktarını değil aynı zamanda adillliğini ve temsil edilebilirliğini de değerlendirmesi gerekir (Dwarakanath et al., 2018).

- **Makine Öğrenimi ve Yazılım Testi Arasındaki Büyük Ayrım:** Geleneksel olarak makine öğrenimi ve yazılım testinin teste yönelik farklı yaklaşımları vardır. Makine öğrenimi, eğitim yoluyla modelin doğruluğunu iyileştirmeye odaklanırken, yazılım testi, beklenmeyenler de dahil olmak üzere çeşitli girdilerle sistemin nasıl performans gösterdiğine bakar. AI sistemlerini etkili bir şekilde test etmek için bu yaklaşımların bir karışımına ihtiyacımız var. Bu, yapay zekanın doğruluğunun çeşitli veri kümeleri üzerinde test edilmesini ve ayrıca gerçek dünya kullanımında karşılaşılabileceği olağandışı veya geçersiz girdileri nasıl ele aldığını kontrol etmeyi içerebilir (Marijan et al., 2019).

- **"Doğru" Yanıtların Zorluğu: Her Zaman Bir Cevap Olmadığında:** Geleneksel yazılım testlerinde, genellikle belirli bir girdi için beklenen çıktıyı tanımlayabiliriz. Örneğin adınızı bir forma girerseniz sistemin adınızı doğru görüntülemesini beklersiniz. Ancak AI sistemlerinde beklenen çıktı daha

belirsiz olabilir. Haber makaleleri yazmak için tasarlanmış bir yapay zeka hayal edin. Belirli bir konu için tek bir "doğru" çıktı olmayabilir. Bu, yapay zekanın doğru çalışıp çalışmadığından emin olmayı zorlaştırıyor. Bu zorluğun üstesinden gelmek için test uzmanları diferansiyel test veya metamorfik test gibi özel teknikler kullanabilir. Bu teknikler, yapay zekanın çıktısını benzer bir sistemle karşılaştırmayı veya potansiyel hataları tanımlamak için çıktıda belirli kalıpları aramayı içerir (Khokhar et al., 2020).

- **Doğru Girdileri Bulmak: Bir Sistemi Sonsuz Olasılıklarla Test Etmek:** Bir AI sisteminin işleyebileceği olası girdilerin aralığı çok geniş olabilir. Kendi kendine giden bir araba düşünün. Otomobilin sonsuz sayıda olası trafik senaryosuna, yol koşullarına ve yaya davranışlarına uygun şekilde tepki verebilmesi gerekiyor. Böyle bir sistemi test etmek için doğru girdi setini seçmek büyük bir zorluktur. Test uzmanları, yapay zekanın yeteneklerini kapsamlı bir şekilde test edebilecek temsili bir girdi örneğini stratejik olarak seçmek için kapsam metrikleri veya kombinatoriyal testler gibi teknikleri kullanabilir (Marijan et al., 2019).
- **Beyaz Kutu ve Kara Kutu Testi: İç Çalışmalar Gizemli Olduğunda:** Beyaz kutu testi, bir sistemin iç işleyişinin incelenmesini içerir. Karmaşık algoritmaların iç işleyişinin anlaşılması ve analiz edilmesi zor olabileceğinden, AI sistemlerinde bu son derece karmaşık olabilir. Sistemin farklı girdilerle nasıl davrandığına odaklanan kara kutu testi genellikle yapay zeka testi için daha pratiktir. Buradaki tekniklerden biri, yapay zekanın nasıl tepki vereceğini görmek için beklenmedik veya rakip girdileri beslemeyi içeren çekişmeli testtir. Bu, yapay zekadaki tipik test yöntemleriyle ortaya çıkarılamayacak zayıflıkların belirlenmesine yardımcı olabilir (Aggarwal, A., et al., 2021).
- **AI Etiği: Adil ve Sorumlu Gelişimin Sağlanması:** Yapay zeka sistemlerinin benzersiz bir zorluğu, onların adil ve tarafsız olmasını sağlamaktır. Daha önce de belirtildiği gibi, önyargılı eğitim verileri yapay zekanın ayrımcı veya adil olmayan kararlar almasına yol açabilir (Johnson, 2018). Adil olmanın

ötesinde, konu AI olduğunda daha geniş etik hususlar vardır. Örneğin, ceza adaleti sisteminde kullanılan bir yapay zeka, üzerinde eğitim aldığı veriler önyargılıysa ırksal önyargıları sürdürebilir.

Bu endişeleri gidermek için AI sistemlerinde adaleti, şeffaflığı ve sorumlu gelişimi teşvik edecek yeni standartlar ve araçlar geliştirilmektedir (Johnson, 2018).

Sonuç olarak AI sistemlerini test etmek dikkatli bir dengeleme eylemi gerektirir. Yapay zekanın doğru, güvenilir ve tarafsız olmasını sağlamamız ve aynı zamanda bu akıllı sistemlerin sunduğu benzersiz zorlukları da göz önünde bulundurmamız gerekiyor. Bu zorlukları anlayarak ve etkili test yöntemleri geliştirerek yapay zekaya güven oluşturabilir ve onun topluma sorumlu ve etik bir şekilde fayda sağlamasını sağlayabiliriz.

3.3.5. Yapay Zeka Testinin Geleneksel Testlere Etkisi

Bu bölüm, yazılım testinin farklı yönlerinin yapay zeka odaklı testlerden ve ilgili araştırmalardan nasıl etkilendiğini araştırıyor. ISTQB'nin sertifikasyon materyallerinde belirtilen terminolojiyi takip ederek geleneksel test uygulamalarına da göz atacağız (ISTQB, 2018).

3.3.5.1. Test Yaklaşımı

Yapay zekanın kullanıma sunulmasıyla birlikte yazılım testine yaklaşımımız çarpıcı biçimde değişiyor. Geleneksel olarak kullanılan üç ana test yöntemine ve yapay zekanın bunları nasıl yenilediğine bir bakalım:

- **Tasarım-Tabanlı Test: Manuel Planlardan AI-Destekli Yardımcılara:** Bu yöntem, yazılım testinin temel dayanağı olmuştur. Test uzmanları, yazılımın karmaşık planlarına dayanarak test senaryolarını titizlikle tasarlar. AI bu planları alıp makine tarafından okunabilir hale getirebilir. Bu, yapay zekanın potansiyel olarak yazılımın yapısını anlamasına ve hatta test senaryolarının oluşturulmasına yardımcı olmasına olanak tanır. Planı okuyan ve farklı özellikleri ve olası

sorunları kapsayacak testler öneren bir AI hayal edin. Bu, test uzmanlarına zaman kazandırır ve daha kapsamlı bir test seti oluşturur.

- **Risk-Esaslı Test: Maksimum Etki için Daha Akıllı Önceliklendirme:** Bu yaklaşım, sınırlı test kaynaklarından en iyi şekilde yararlanmaya odaklanır. Geleneksel olarak test uzmanları, deneyimlerine ve potansiyel riskleri anlamalarına dayanarak alanlara öncelik verir. Tahmine Dayalı Test Önceliklendirme (PTP) gibi makine öğrenimi teknikleri bu sürece zeka katıyor. PTP, ilk önce çalıştırılacak en kritik testleri belirlemek için kodun karmaşıklığı, geçmiş sorunlar ve kullanıcı davranışı gibi şeyleri analiz eder. Bu, test uzmanlarının büyük hataları erkenden yakalamasını ve daha sonraki gecikmeleri önlemesini sağlar (Chen, et al., 2018).
- **Model Tabanlı Test (MBT): Test için Sanal Bir Kopya Oluşturma:** Oluşturduğunuz yazılımın dijital bir ikizini, onun davranışını taklit eden sanal bir kopyasını düşünün. MBT'nin özü budur. İlginç bir şekilde AI test çözümleri büyük ölçüde MBT'ye dayanıyor. İşleyiş şekli şu şekildedir: AI, yazılımı analiz eder ve sanal bir model oluşturur. Bu model daha sonra test senaryoları oluşturmak için kullanılır. (Gerrard P, 2017)'nin işaret ettiği gibi, ister resmi bir model olsun, ister testçinin kafasında oluşturduğu zihinsel model olsun, tüm testler bir tür model içerir. MBT, tüm test uzmanlarının kullanabileceği merkezi, paylaşılan bir model oluşturarak bunu bir adım daha ileri götürüyor. Bu, işbirliğini teşvik eder, test stratejilerinde tutarlılık sağlar ve sonuçta daha kapsamlı bir yazılım değerlendirmesine yol açar.

Bunlar yapay zekanın yazılım testini nasıl dönüştürdüğünün sadece birkaç örneğidir. Bu yeni yapay zeka destekli yaklaşımları benimseyerek, daha hızlı ve daha az hatayla sunulan daha güçlü ve daha güvenilir yazılım uygulamaları oluşturabiliriz.

Yazılım testinin geleceği, yüksek kaliteli bir yazılım dünyası oluşturmak için birlikte çalışan insan uzmanlığı ve yapay zekanın güçlü bir birleşimidir.

3.3.5.2. Test Teknikleri

Yazılım testi, denemek için rastgele eylemler seçmenin ötesine geçer. Test uzmanları, doğru testleri tasarlamak ve seçmek için 'test teknikleri' olarak bilinen belirli yöntemler kullanır. Yapay zekanın bu teknikleri nasıl değiştirdiği aşağıda açıklanmıştır:

- **Planı Takip Etmek mi Yoksa Kendi Başına Keşfetmek mi? Spesifikasyon ve Keşif Karşılaştırması:** Geleneksel olarak iki ana test tekniği türü vardır:
 - **Spesifikasyona dayalı (kara kutu):** Bu teknikler, testleri tasarlamak için yazılımın planına (spesifikasyonlarına) dayanır. Örneğin, "sınır değer analizi" adı verilen bir teknik, yazılımın beklenen girdilerin sınırlarında nasıl davrandığını kontrol eder. Yapay zeka potansiyel olarak bu spesifikasyonları analiz edecek ve hatta bunlara dayalı test senaryoları oluşturacak şekilde eğitilebilir.
 - **Yapı tabanlı (beyaz kutu):** Bu teknikler, testleri tasarlamak için yazılımın iç yapısına bakar.
- **Yapay Zeka Destekli Keşif: İnsan Test Cihazı Gibi Öğrenme:** "Keşif testi" gibi teknikler, test uzmanlarının yazılımı özgürce keşfetmesini ve hataları bulmak için farklı şeyler denemesini içerir. Yapay zeka testi, yapay zekanın tıpkı bir insan testçinin yapacağı gibi çeşitli eylemleri deneyerek yazılımı nasıl kullanacağını "öğrendiği" bir keşif testi biçimi olarak görülebilir. Yapay zeka daha sonra etkileşimli öğeleri tanımlayabilir ve farklı eylemler gerçekleştirdiğinde ne olacağını test edebilir. Bu yaklaşım bölüm 3.5.5.1'de daha ayrıntılı olarak açıklanmaktadır (Meinke, et al., 2011).

Geleneksel olarak test uzmanları proje ihtiyaçlarına göre belirli tekniği seçerler. Gelecekte AI, test edilen yazılıma göre en uygun tekniği otomatik olarak önerebilir.

Bu, yapay zekanın test teknikleri dünyasını nasıl sarstığına dair sadece bir bakış. Geleneksel yöntemleri yapay zekanın gücüyle birleştirerek, hataları bulma ve düzeltmenin daha etkili ve verimli yollarını oluşturabilir ve böylece daha yüksek kaliteli yazılımlar elde edebiliriz.

Teknik Türü	Açıklama	AI'nin Nasıl Geliştirebileceği
Şartname Tabanlı (Kara Kutu)	Program şartnamelerine dayalı testler tasarlar, örneğin "sınır değeri analizi"	AI, bu şartnameleri analiz etmek ve bunlara dayalı test senaryoları oluşturmak için eğitilebilir
Yapı Tabanlı (Beyaz Kutu)	Programın iç yapısına dayalı testler tasarlar	
Keşifsel Test	Testçiler programı serbestçe keşfederek hataları bulur	AI, insan testçilere benzer şekilde farklı prosedürleri deneyerek "öğrenebilir"

Tablo 1: Test Teknikleri ve AI Geliştirmeleri

3.3.5.3. Yazılım Test Otomasyonu

Yazılım testi uzun süredir test otomasyonuna dayanıyordu; testleri tekrar tekrar çalıştırabilen bilgisayar programları oluşturarak test uzmanlarının değerli zaman ve çabalarından tasarruf etmesini sağladı. Geleneksel olarak bu testler, programın yapması gereken her eylemi ve beklenen sonuçları titizlikle detaylandıran önceden yazılmış senaryolara dayanır (Daniel & Bruce, 2002). Etkili olmasına rağmen bu komut dosyaları katıdır ve yazılımdaki değişikliklere uyum sağlamakta zorlanır. Yapay zekanın devreye girdiği yer burasıdır ve test otomasyonu dünyasına bir zeka ve esneklik dalgası getirir.

Yapay Zekanın Öğrenme Avantajı: Katı Komut Dosyalarının Ötesinde, Uyarlanabilir Testlere Doğru: Yapay zekanın en önemli güçlü yönlerinden biri, öğrenme ve uyum sağlama yeteneğidir. Geleneksel komut dosyalarının aksine, yapay zeka destekli test otomasyon araçları deneyimlerden öğrenebilir ve yazılımın davranışına göre yaklaşımlarını ayarlayabilir. Bu, yazılı bir testi aksatabilecek beklenmedik durumlarla başa çıkmalarına olanak tanıyarak onları daha sağlam ve güvenilir hale getirir. Ekrandaki belirli bir düğmeye tıklamak için tasarlanmış bir test komut dosyası düşünün. Yeni bir yazılım güncellemesinde düğmenin konumu biraz değişirse komut dosyası başarısız olabilir. Ancak AI destekli bir araç, işlevine veya görsel öğelerine göre düğmeyi

tanıyabilir ve eylemlerini buna göre uyarlayarak testin sorunsuz bir şekilde devam etmesini sağlayabilir.

Anahtar Kelimeler: İnsanlar ve AI Arasında Bir Köprü Kurmak: Test otomasyonunda yapay zekanın gücünden yararlanmaya yönelik yaklaşımlardan biri "anahtar kelime odaklı testi" içerir. Bu teknik, test adımlarını ("Giriş" veya "ClickButton" gibi) temel teknik ayrıntılardan ayırır. Bu iş bölümü, test uzmanlarının test etmek istedikleri şeye, yani anahtar kelimelerin temsil ettiği üst düzey işlemlere odaklanmasına olanak tanır. Yapay zeka, perde arkasında testleri çalıştırmının, anahtar kelimeleri belirli eylemlere dönüştürmenin ve sonuçları doğrulamanın teknik yönleriyle ilgilenir (ISTQB, 2018). Bu yaklaşım, kodlama uzmanı olmayan test uzmanlarına, test otomasyonu için yapay zekanın gücünden yararlanma olanağı sağlar.

Hedef Odaklı Test: Stratejik Test Planlayıcısı Olarak AI: Test otomasyonunda yapay zekanın bir başka heyecan verici uygulaması da "hedef odaklı testtir". Burada test uzmanı, yazılım için istenen sonucun (hedef) ve başlangıç noktasının (başlangıç durumu) ana hatlarını çizen bir strateji uzmanına benzer. Akıllı bir test planlayıcısı olarak hareket eden AI, daha sonra görevi devralır. Uygulama modelini analiz eder, potansiyel yolları belirler ve yazılımı başlangıç durumundan istenen sonuca kadar yönlendiren bir dizi test senaryosu oluşturur. Bu teknik aynı zamanda gerektiğinde diğer test yöntemlerini de entegre ederek kapsamlı bir test paketinin oluşturulmasını sağlayabilir (Memon, et al., 1999).

Test hedeflerini tanımlamak, sonuçları analiz etmek ve genel rehberlik sağlamak için insan uzmanlığı hayati önem taşımaya devam ederken AI, testleri otomatikleştirme şeklimizde devrim yaratıyor.

Bu yeni AI destekli araçlar önemli avantajlar sunuyor:

- **Karmaşık Senaryoların Ele Alınması:** AI, manuel olarak yazılması zor veya zaman alıcı olabilecek karmaşık test senaryolarının üstesinden gelebilir. Bu, geleneksel yöntemlerden kaçabilecek potansiyel hataları tespit ederek daha kapsamlı testlere olanak tanır.

- **Değişime Uyum Sağlamak:** Yazılımın sürekli gelişen doğası, geleneksel komut dosyalarını geçersiz kılabilir. Yapay zekanın öğrenme ve uyum sağlama yeteneği, yazılımda değişiklikler olsa bile testlerin güncel kalmasını sağlar.
- **Test Uzmanlarına Serbest Bırakma:** AI, tekrarlanan görevleri otomatikleştirerek ve karmaşık senaryoları yöneterek, insan test uzmanlarına değerli zaman kazandırır. Bu onların test stratejileri tasarlama, sonuçları analiz etme ve hataları düzeltmek için geliştiricilerle işbirliği yapma gibi daha üst düzey görevlere odaklanmalarına olanak tanır.

İnsan uzmanlığı ile AI arasındaki sinerji, yazılım testi otomasyonunda yeni bir verimlilik ve etkinlik çağına yol açıyor. Ağır yükün üstesinden gelen AI destekli araçlar sayesinde, test uzmanları stratejik güçlü yönlerine odaklanabilir ve sonuçta daha kaliteli yazılım uygulamalarının sunulmasına yol açabilir.

3.3.5.4 Regresyon Testi

Regresyon testi, yazılım geliştirmede çift kontrol sürecidir. Değişiklikler yapıldıktan sonra programda testlerin tekrar çalıştırılmasını içerir. Bu, söz konusu değişikliklerin yeni sorunlara yol açmamasını veya daha önce çalışan herhangi bir şeyi bozmamasını sağlar (ISTQB, 2018). Her şeyi sıfırdan tekrar test etmenin sınırlı zaman ve kaynaklar nedeniyle gerçekçi olmayabileceği karmaşık yazılım sistemleri için özellikle önemlidir. Regresyon testini daha akıllı hale getirmek için yapay zekanın devreye girdiği yer burasıdır:

- **En Önemli Şeye Odaklanmak: Testlere Öncelik Vermek:** Geleneksel olarak, test uzmanlarının değişiklik yapıldıktan sonra birçok testi yeniden çalıştırması gerekebilir. Bu yavaş ve verimsiz olabilir. Tahmine Dayalı Test Önceliklendirmesi gibi AI teknikleri, hangi testlerin ilk önce çalıştırılmasının en önemli olduğunu belirlemenize yardımcı olabilir. PTP, değiştirilen kodun ne kadar karmaşık olduğu, geçmişte karşılaşılan sorunlar ve kullanıcıların yazılımla genel olarak nasıl etkileşim kurduğu gibi çeşitli faktörleri dikkate alır. Bu şekilde, test uzmanları değişikliklerden etkilenmesi en muhtemel alanlara odaklanarak değerli zaman ve çabadan tasarruf edebilirler (Chen, et al., 2018).

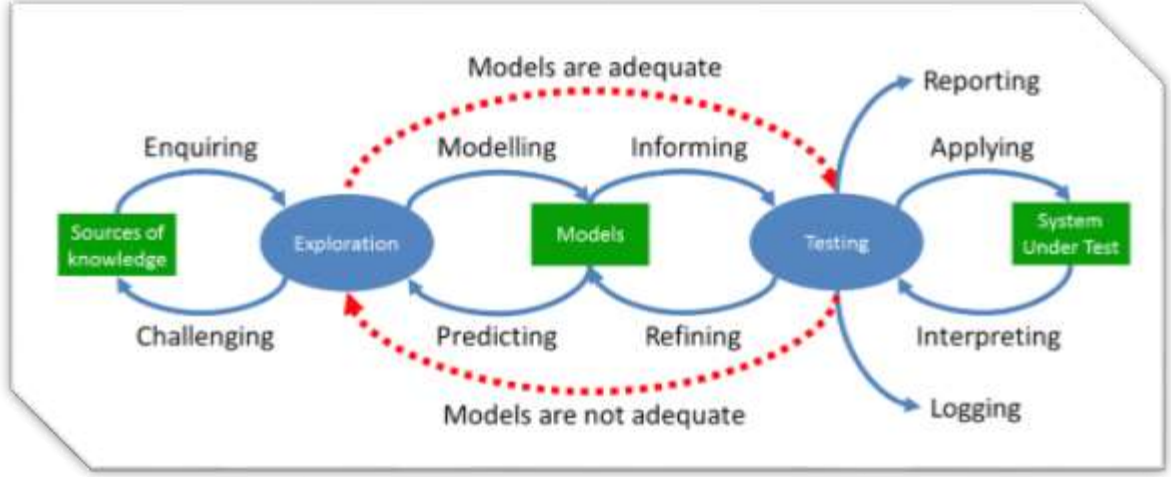
- **Akıllı Seçim ve Minimizasyon:** Önceliklendirmenin yanı sıra, regresyon testi için gereken test sayısını azaltmanın başka yolları da vardır. Bu teknikler, yapılan kesin değişikliklere dayalı olarak yalnızca belirli testlerin seçilmesini veya bir test senaryosu içindeki adımların kolaylaştırılmasını içerebilir. Yapay zeka henüz bu alanlarda bu kadar yaygın kullanılmasa da gelecekteki ilerleme potansiyeline sahip, devam eden bir araştırma alanıdır (Remo L, 2018).

AI, özünde, test uzmanlarının ilk önce çalıştırılacak en önemli testleri belirlemesine yardımcı olarak regresyon testini dönüştürüyor. Bu daha verimli ve etkili bir sürece yol açar. Sonuç olarak geliştiriciler, yüksek yazılım kalitesini korurken yeni özellikleri ve hata düzeltmelerini daha hızlı yayınlayabilir.

3.4. “Yazılım Testi için Yeni Model” – Yapay Zeka Testini Oluşturma Modeli

Yazılım testi dünyasında Paul Gerrard olaylara yeni bir bakış açısı önerdi. Kullandığımız geleneksel yöntemlerin gelecekte yeterli olmayacağına inanıyor. İşte nedeni:

- **Yazılım geliştirme hızlanıyor:** Yazılımın oluşturulma şekli değişiyor. Yeni özellikler ve güncellemeler daha sık yayınlanıyor ve test uzmanları üzerinde buna ayak uydurmaları için baskı oluşturuyor.
- **Yazılım daha karmaşık hale geliyor:** Yazılım sistemleri giderek daha karmaşık hale geliyor, bu da onların kapsamlı bir şekilde test edilmesini daha zor hale getiriyor.
- **Test sorumluluğu paylaşıyor:** Test sorumluluğu artık yalnızca test uzmanlarının omuzlarında değil. Geliştiriciler ayrıca test sürecinde daha büyük bir rol üstleniyorlar.



Şekil 3 - New Model for Testing (Gerrard P, 2020)

Gerrard'ın Şekil 3'te gösterilen modeli ideal bir test uzmanının zihniyetini temsil etmektedir. Kullanılan spesifik araçlara veya süreçlere bakılmaksızın uygulanabilecek genel bir çerçevedir. Model, yerleşik test en iyi uygulamalarını ve test uzmanlarının işlerine nasıl yaklaşmaları gerektiğini yansıtır. Bu, onu aşağıdaki bölümlerde daha ayrıntılı olarak tartışılan AI-destekli testler için mükemmel bir temel haline getirir (Gerrard P, 2020).

3.5. AI Testi – AI Test Sistemi

AI testi, test sürecini otomatik olarak yürütmek için "akıllı öğrenme agentlarını" kullanan yeni bir yaklaşımdır (Božić, V., 2024). Kendi kendine öğrenebilen ve hareket edebilen bir yazılım programı hayal edin. Bu "agents" test ettikleri yazılımı keşfedebilir, nasıl çalıştığını anlayabilir ve hatta test senaryolarını tasarlayıp çalıştırabilirler. Ayrıca mini test uzmanlarından oluşan bir ekip gibi davranarak birlikte çalışabilir ve birbirleriyle iletişim kurabilirler.

Yapay Zeka Testinin Faydaları: Evrensel Bir Test Araç Kutusu

AI testinin en önemli avantajlarından biri genelleştirilmiş testler oluşturma yeteneğidir. Bu testler yalnızca bir uygulamaya özel değildir; belirli bir alandaki çeşitli benzer uygulamaları test etmek için kullanılabilirler. Bunu ortak kullanıcı eylemlerinden oluşan bir kitaplık gibi düşünün; AI agentsı bir uygulamada (arama kutusu gibi) belirli bir öğeyle

karşılaştığında, kitaplıktaki ilgili test senaryolarını arayabilir ve bu testleri benzer bir aramaya sahip herhangi bir uygulamada çalıştırabilir. Bu, aynı testleri farklı uygulamalar için yeniden yazma ihtiyacını ortadan kaldırarak zamandan ve emekten tasarruf sağlar. Ek olarak, bu testler bir kuruluş içindeki farklı ekipler arasında kolayca paylaşılarak işbirliği teşvik edilebilir. Bulut tabanlı çözümler, katılan herkes için ölçeklenebilirlik ve kolay erişim sunarak bunu daha da kullanışlı hale getiriyor.

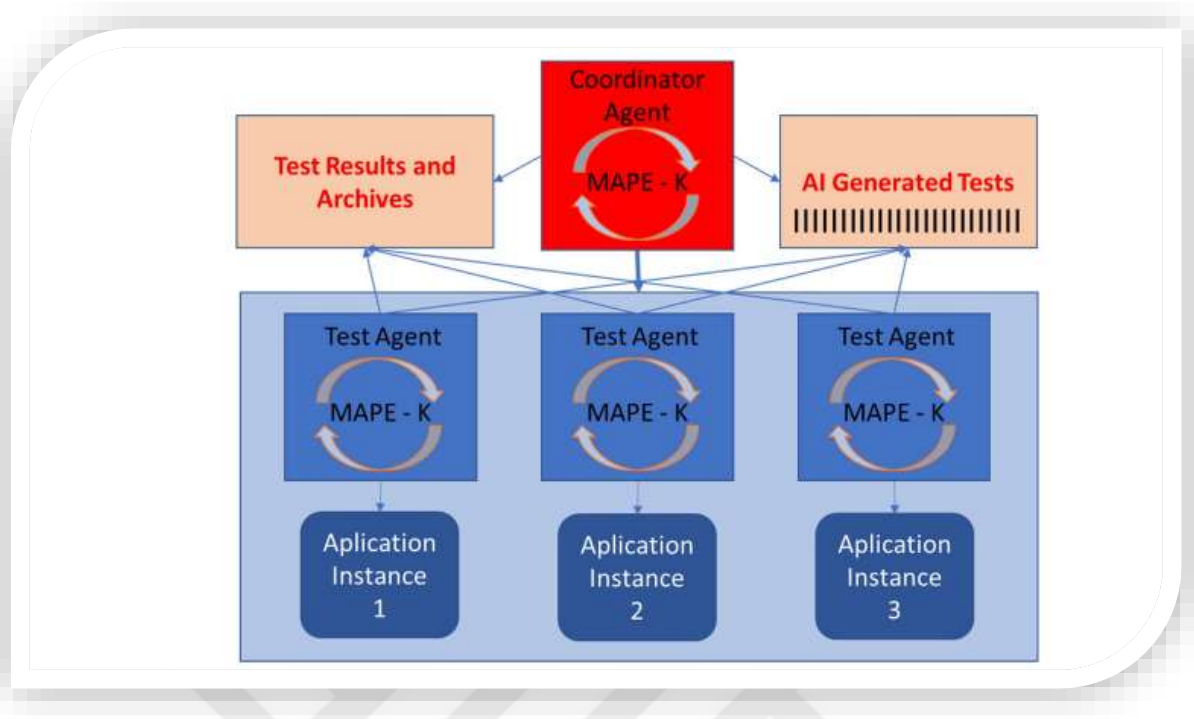
AI Test Çerçevesini Oluşturma: İlham ve Yenilik

Bu AI test çerçevesinin genel tasarımı çeşitli kaynaklara dayanmaktadır. Temel konsept, Tariq'in "Test Dünyası için AI ve ML Becerileri" (Tariq M, 2018) konulu sunumunun yanı sıra Tariq ve Arbon tarafından ortaklaşa oluşturulan slaytlardan gelmektedir. Tasarım aynı zamanda Paul Gerrard'ın "Yazılım Testi için Yeni Model" (Gerrard P, 2017) konseptinden esinlenerek yazarın kendi fikirlerini ve iyileştirmelerini de içermektedir. Mevcut bilgi ve yeni fikirlerin birleşimi, geleceğe yönelik bir AI test çerçevesinin yolunu açıyor.

3.5.1. AI Agentlar

AI testi, yazılım uygulamalarını keşfetmek ve test etmek için "test agentları" adı verilen bir veya daha fazla özel program kullanır. Bu agentların iki çeşidi vardır:

- **Koordinatör agentsi:** Bu agent, tüm test sürecini denetleyen bir test yöneticisi gibi hareket eder. Testleri takip eder, kopyalardan kurtulur ve görevleri diğer agentlara atar.
- **Test agentsı:** Bu agentlar, test edilen yazılımla doğrudan etkileşime giren araçlardır. Nasıl çalıştığını anlamak için uygulamayı keşfederler ve bulduklarına göre test senaryoları oluşturup çalıştırırlar.



Şekil 4: Functional Environment of AI Test Agents (Tariq M, 2018)

Yaparak Öğrenme: AI Test Agentları Zaman İçinde Nasıl İyileşir

Bu AI agentları "MAPE-K kontrol döngüsü" adı verilen bir kavrama dayanmaktadır. Dört adımdan oluşan bir döngü hayal edin: İzle, Analiz Et, Planla ve Bil ("K" bilgi anlamına gelir). Bu döngü esas olarak agentların zaman içinde nasıl öğrendiğini ve geliştiğini temsil eder.

- **Monitör:** agentlar, test ettikleri yazılımı sürekli olarak gözlemleyerek olup biten her şeyi takip ederler.
- **Analiz Et:** agentlar topladıkları bilgileri alır ve ne anlama geldiğini anlamak için analiz eder.
- **Planla:** agentlar analizlerine dayanarak bundan sonra ne yapacaklarını planlarlar. Bu, yeni testler oluşturmayı, mevcut testleri çalıştırmayı veya yazılımın farklı bölümlerini keşfetmeyi içerebilir.
- **Bilin:** agentlar öğrendikleri her şeyi bir bilgi tabanında saklarlar. agentlar daha fazla bilgi topladıkça bu bilgi tabanı sürekli olarak büyüyor ve gelişiyor. Bu bilgiyi diğer agentlarla de paylaşabilirler.

Bağımsız ve Birlikte Çalışmak: Yapay Zeka Testi Nasıl Düzenlenir

İdeal olarak her test agentsi diğerlerine müdahale etmeden kendi başına çalışabilmelidir. Bunu başarmanın birkaç yolu vardır:

- **Böl ve yönet:** Koordinatör agent, yazılımın farklı özelliklerini farklı test agentlarına atayabilir. Bu şekilde her agent diğerlerinin yoluna girmeden kendi alanına odaklanabilir.
- **Konteyner teknolojisi:** Test edilen her yazılım parçası için ayrı bir kutu düşünün. Konteyner teknolojisinin yaptığı da budur. Her test agentsı, kapları kullanarak yazılımın kendi yalıtılmış kopyası üzerinde çalışabilir ve birbirlerini etkilememelerini sağlayabilir.

Bulut Gücü: AI Testini Artırma

Bulut bilişim, AI test sistemlerinin gerektiği gibi büyüyüp küçülmesine olanak tanır. Bu, büyük veya karmaşık yazılım sistemlerini yönetmek için kolayca daha fazla kaynak (işlem gücü gibi) ekleyebileceğiniz anlamına gelir. Bulutun yapay zeka testlerine nasıl yardımcı olduğu aşağıda açıklanmıştır:

- **Daha fazla test çalıştırın:** Daha fazla kaynakla aynı anda daha fazla test çalıştırabilirsiniz.
- **Daha fazla veri oluşturun:** Testler genellikle çok fazla veri üretmeyi içerir. Bulut, bu veriler için gereken depolama alanını sağlar.
- **Test ortamlarının kurulması:** Bulut tabanlı araçlar, test için gereken ortamları hızlı ve kolay bir şekilde kurmamıza yardımcı olabilir.

AI Test Süreci: Adım Adım Bir Bakış

AI test sürecinde dört ana adım vardır:

1. **Keşfedin ve Öğrenin:** Test agentleri ilk önce yazılımın nasıl çalıştığına dair temel bir anlayışa sahip olmak için yazılımı keşfederler.

2. **Bir Model Oluşturun:** agentlar, keşiflerine dayanarak yazılımın bir modelini oluşturur. Bu model, agentların yazılımın yapısını ve nasıl çalıştığını anlamalarına yardımcı olan bir harita gibidir.
3. **Testleri Çalıştırın:** agentlar, test senaryolarını oluşturmak ve çalıştırmak için modeli kullanır. Bu test senaryoları, yazılımın beklendiği gibi çalışıp çalışmadığını kontrol etmek için tasarlanmıştır.
4. **Öğrenin ve Geliştirin:** agentlar, yaptıkları testlerin sonuçlarından sürekli olarak öğrenirler. Bu bilgileri yazılım modellerini güncellemek ve test senaryolarını geliştirmek için kullanırlar.

AI test agentları, sürekli öğrenerek ve uyum sağlayarak hataları bulma ve yazılımın düzgün çalışmasını sağlama konusunda giderek daha etkili hale gelebilir.

3.5.2. Keşfetmek ve Algılamak

AI testinde ilk adım, test etme agentlarının yazılım uygulamasını keşfetmesi ve anlamasıdır. Bu, yazılımın farklı bölümlerinin ne yaptığını ve kullanıcıların bunlarla nasıl etkileşime girebileceğini bulmayı içerir. İşte bu sürecin bir dökümü:

- **Kullanıcı Arayüzü (UI) Öğelerini Sınıflandırma:** AI agentları, düğmeler, menüler ve metin kutuları gibi farklı öğeleri belirlemek için yazılımın kullanıcı arayüzünü (kullanıcıların etkileşim kurduğu ekranlar ve düğmeler) inceler. Esasen, şu soruyu sorarlar: "Kullanıcılar bu uygulama ile gerçekten ne yapabilir?"
- **Kullanıcı Arayüzünün Ötesine Geçmek:** AI testi kullanıcı arayüzlerini analiz edebilirken, bununla sınırlı değildir. Ayrıca, birim testi (yazılımın ayrı bölümlerini test etme) veya hatta arka planda çalışan hizmetler gibi kullanıcı arayüzü olmayan uygulamaları test etme gibi diğer test türleri için de kullanılabilir. Yazılımın girdi alma şekli (kullanıcı tıklamaları veya diğer uygulamalardan gelen veriler gibi) AI agentlarının onu nasıl keşfettiğini etkileyebilir, ancak temel ilkeler aynı kalır.
- **Kontrol Noktalarını Belirleme:** agentlar uygulamayı keşfederken, düğmeler veya metin alanları gibi etkileşim kurabilecekleri şeyleri ararlar. Bunu yapmak için yazılımın dahili kodunu analiz etme, kullanıcı arayüzünün oluşturulma

şeklini inceleme ve hatta görüntü tanıma kullanma gibi bir dizi tekniği bir arada kullanabilirler.

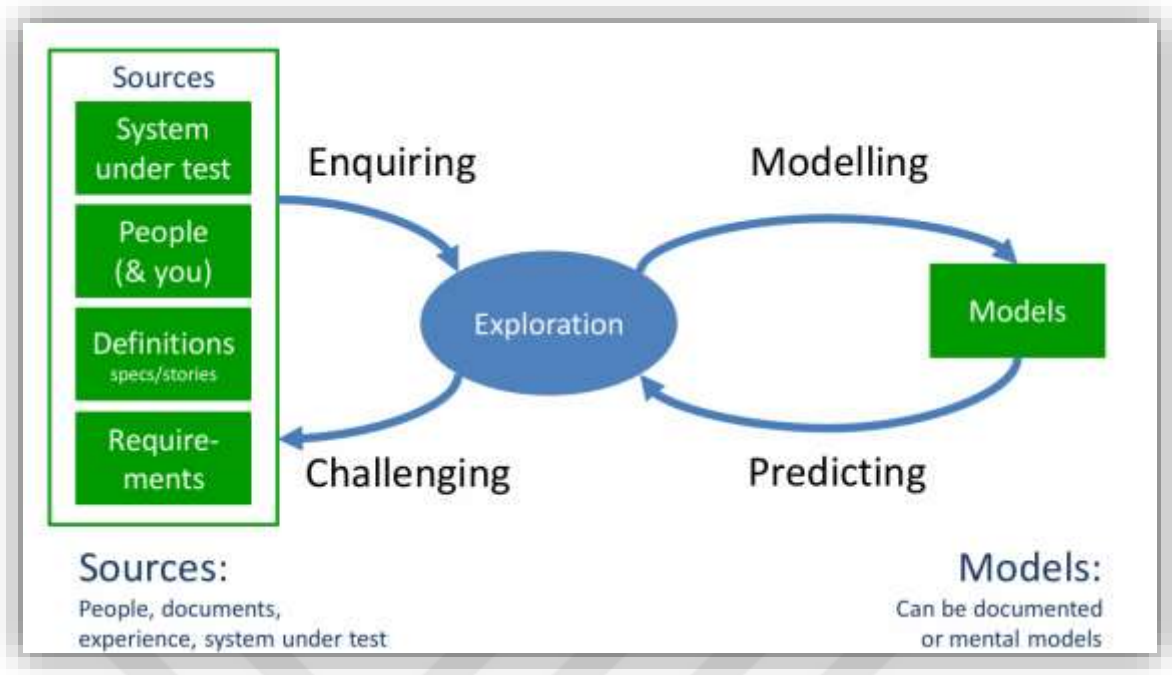
- **Geçmiş Keşiflerden Öğrenme:** Zamanla, AI agentları yazılımı birden fazla test turunda keşfettikçe, farklı sürümleri karşılaştırabilir ve değişiklikleri belirleyebilirler. Bu, yazılımın nasıl geliştiğini anlamalarına yardımcı olur.

Zorluklar ve İlham: İnsanların AI Testçilerinin Keşfetmesine Nasıl Yardımcı Olduğu

AI agentları kendi başlarına yazılım keşfetmede daha iyi hale gelirken, hala zorluklar var. Örneğin, bir yazılım parçası için gereklilikleri ve özellikleri nasıl anlayacağını bulmak AI için zor olabilir. Ancak, buna yardımcı olmak için geliştirilen araçlar var.

Geleneksel olarak, insan testçiler yazılımın kendisi, geliştiricilerle yapılan görüşmeler ve yazılımın özelliklerini ve gereksinimlerini ana hatlarıyla belirten belgeleri incelemek de dahil olmak üzere yazılım hakkında mevcut tüm bilgi kaynaklarını araştırır. Yapay zeka için zorluk, bu tür bilgileri işlemenin yollarını bulmaktır. Neyse ki, bu alanda araştırmalar devam ediyor ve boşluğu kapatmak için yeni araçlar yaratılıyor.

Metinde, uygulamanın kendisini keşfetmeye odaklanan Tariq'in bir sunumundan ve daha geniş bir bilgi kaynağı yelpazesini içeren Paul Gerrard'ın bir modelinden bahsediliyor. Bu farklı yaklaşımlar, kapsamlı bir AI test stratejisi geliştirmek için değerli içgörüler sağlar.



Şekil 5 - New Model for Software Testing - Exploration Process (Gerrard P, 2017)

3.5.3. Modelleme

AI testleri "model-tabanlı test" adı verilen bir teknik kullanır. Bu, AI agentlarının test ettikleri yazılım uygulamasının bir tür haritasını oluşturduğu anlamına gelir. Tıpkı biz insanların bir programın nasıl çalıştığına dair zihinsel bir resim çizdiğimiz gibi, bu AI agentları da kendi yollarıyla bir model oluştururlar.

Metinde, insanların modelleri nasıl oluşturduğunu gösteren bir şekil (Şekil 5) belirtilir; test planları yazabilir veya belirli bir plan olmadan yazılımı özgürce incelediğimiz keşifsel test yöntemlerini kullanabiliriz. Hangi yöntem kullanılırsa kullanılsın, test edenlerin (hem insan hem de AI) test çabalarını yönlendirecek bir tür modele ihtiyacı vardır.

Bu modelleri temsil etmenin farklı yolları vardır, örneğin durum makineleri veya Markov zincirleri kullanmak. Önemli olan, test edilen yazılımı ve test sürecinin hedeflerini doğru bir şekilde yansıtan bir model seçmektir.

Model Oluşturma ve Karmaşıklıkla Başa Çıkma

Model tabanlı testle ilgili zorluklardan biri "giriş alanı patlaması" olarak adlandırılan bir şeydir. Tıklayabileceğiniz veya etkileşimde bulunabileceğiniz 100 farklı şeyin olduğu bir web sayfasını hayal edin ve diyelim ki bu sayfayı kullanmanın tipik bir yolu 30 adım içeriyor. Bu, yazılımda 100'ün 30'un kuvvetine yükseltilmiş olası yollar yaratır ki bu da modelleme için çok büyük bir olasılık sayısıdır!

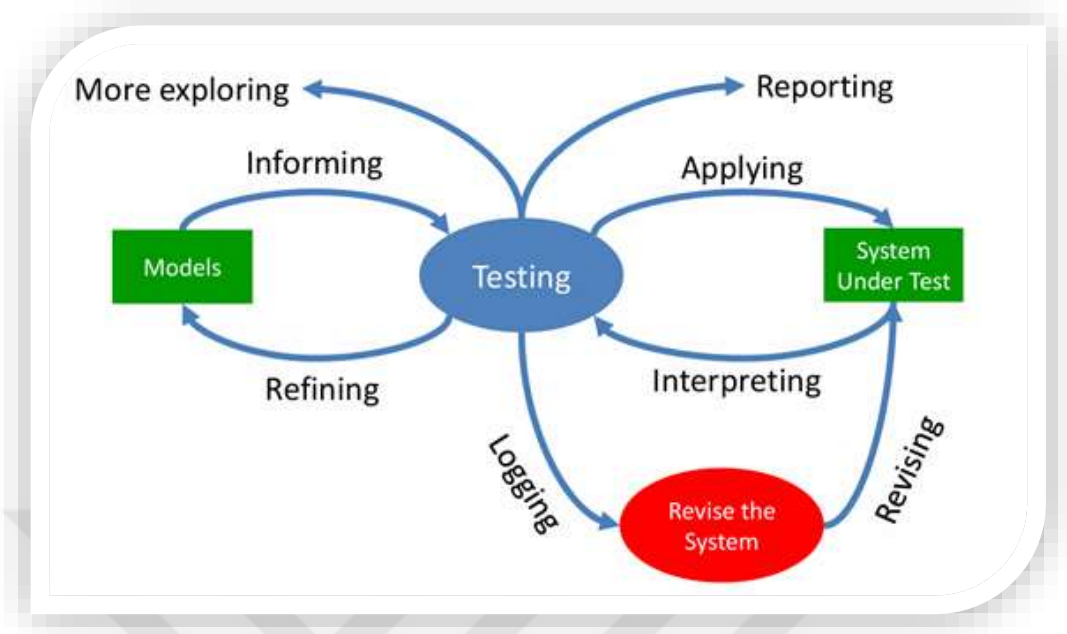
Bu karmaşıklıkla başa çıkmak için, AI testi "olasılıksal modelleme" adı verilen bir teknik kullanabilir. Temel fikir, yazılımın bazı bölümlerinin test edilmesinin diğerlerinden daha önemli veya ilginç olmasıdır. AI agentları en önemli alanlara odaklanarak daha yönetilebilir bir model oluşturabilir.

Metinde, olasılıksal modelleme hakkında daha ayrıntılı bilgi veren Tariq'in bir sunumundan bahsediliyor. Bu teknik, AI testinin en kritik alanlara odaklanmasına ve yazılım içindeki olası etkileşimlerin çokluğu nedeniyle bunalmamasına yardımcı olur.

3.5.4. Etkileşim

AI agentları yazılım uygulamasının bir modelini oluşturduktan sonra, bunu yazılımla etkileşim kurmak ve işlevselliğini test etmek için kullanabilirler. Bu şu şekilde çalışır:

- **Modeli Kılavuz Olarak Kullanma:** AI agentları, yazılımda gezinmek için oluşturdukları modeli bir yol haritası olarak kullanırlar. Bunu, hangi eylemleri gerçekleştireceklerini, hangi verileri gireceklerini ve hangi sonuçları bekleyeceklerini anlamak için kullanabilirler.
- **Hedef -Tabanlı Test:** Yapay zeka testinde kullanılan bir yaklaşıma "hedef tabanlı test" denir. Bu test türünde, yapay zeka agentlarına belirli bir hedef verilir ("uygulamaya giriş yapmak" gibi) ve bu hedefe ulaşmak için modellerini kullanırlar. Eylemlerini yönlendirmek için test planlarından gelen üst düzey talimatları kullanabilirler. Ayrıca, bu etkileşimleri otomatikleştirmeye yardımcı olabilecek Selenium WebDriver gibi araçlar da mevcuttur.



Şekil 6 - New Model for Software Testing Testing Process (Gerrard P, 2017)

Metinde bu etkileşim aşamasını gösteren Gerrard'ın bir figüründen (Şekil 6) bahsediliyor.

- **Sonuçları Yorumlamanın Zorluğu:** "Test oracle" sorununu çözmeye yönelik girişimler olsa da (ki bu esasen test sonuçlarının doğru olup olmadığını anlamak anlamına gelir), test sonuçlarını yorumlamak hala çoğunlukla insanlar tarafından yapılmaktadır. Beklenen sonuçların ne olduğuna karar vermek ve modelleri gerçek sonuçlara göre güncellemek için insanlara ihtiyaç vardır. İdeal olarak, bunu yapmanın daha otomatik bir yolu olurdu; belki de modelleri oluşturmak için yazılım gereksinimlerini ve özelliklerini kullanarak veya bölüm 3.6.3.6'da belirtilen teknikleri kullanarak.
- **Modeli İyileştirme:** Sonuçları test etme ve yorumlama süreci, AI agentlarının yazılım modellerini iyileştirmelerine yardımcı olur. Bu, bir sonraki bölümde daha ayrıntılı olarak tartışacağımız devam eden bir öğrenme ve keşfetme sürecidir.

3.5.5. Öğrenme

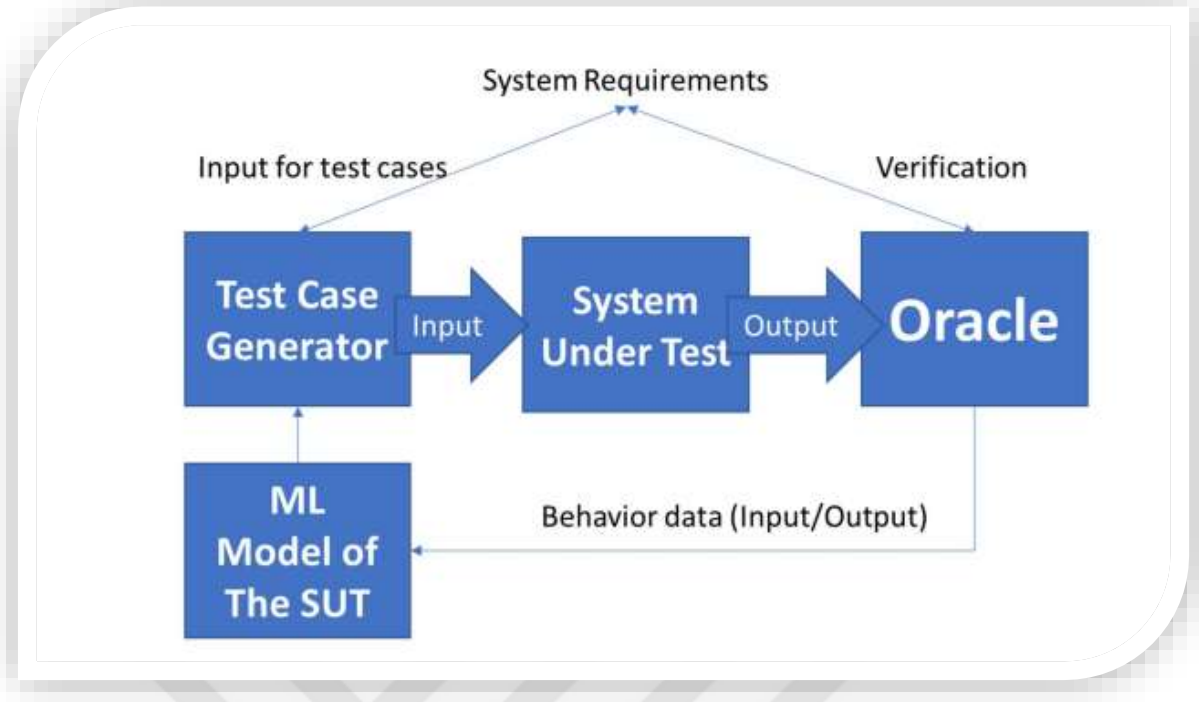
AI test agentları, yazılımla etkileşimlerinin sonuçlarından öğrenerek zamanla daha akıllı hale gelir. İşte nasıl çalıştığı:

- **Basit Başlamak:** AI agentları yazılımla ilk etkileşime girdiğinde, eylemleri rastgele veya hatta anlamsız görünebilir. Bunun nedeni, yazılımın nasıl çalıştığını öğrenmenin henüz erken aşamalarında olmalarıdır.
- **Her Testte Öğrenme:** AI agentları daha fazla test yaptıkça, giderek daha karmaşık hale gelirler. Etkileşimlerinin sonuçlarından öğrenirler ve bu bilgileri gelecekteki eylemlerini iyileştirmek için kullanırlar.
- **Beklenmeyen Sonuçların Değeri:** agentlar her testte kendileri için belirlenen belirli hedeflere her zaman ulaşmasalar bile, rastgele etkileşimleri bazen beklenmedik sonuçlara yol açabilir. Bu beklenmedik sonuçlar değerli olabilir çünkü insan testçilerin gözden kaçırmış olabileceği hataları veya sorunları ortaya çıkarabilirler. Başka bir deyişle, AI agentları her zaman planı takip etmese bile, yine de önemli bilgiler bulabilirler.

Deneme ve yanılma yoluyla bu tür öğrenme, AI testinin önemli bir yönüdür. Agentların hataları bulma ve yazılımın düzgün çalışmasını sağlama becerilerini sürekli olarak geliştirmelerine olanak tanır.

3.5.5.1. Öğrenmeye Dayalı Teknik

Başlangıçta, AI testleri Meinke ve meslektaşları tarafından 2011 yılında tanıtılan Öğrenme Tabanlı Teknik olarak bilinen bir yöntemi benimsedi. (Şekil 7)



Şekil 7- Learning based test technique (Meinke, et al., 2011)

Bu yöntem, spesifikasyonlara dayalı bir test türüdür. Makine öğrenimini kullanarak test vakalarını otomatik olarak oluşturarak geleneksel spesifikasyona dayalı testi geliştirmeyi amaçlar. (Meinke, et al., 2011) Eğitim sürecine daha fazla güvenmek için birden fazla öğrenme yönteminin kullanılması önerilir.

Paul Gerrard'ın teorisine göre, öğrenme keşif aşamasında gerçekleşir. Bu, kişinin bilgi kaynakları hakkında sorular sorduğu zamandır. Bu, modelleri uygulayarak Test Edilen Sistemi (SUT) anlamak, sonuçların ne anlama geldiğini anlamak ve test sonucu bilinmiyorsa ancak kontrol doğru şekilde yapılmışsa modelleri iyileştirmek anlamına gelir.

Yapay zeka, tanımları ve gereksinimleri bir modele nasıl dönüştüreceği zorluğuyla karşı karşıyadır. Bir çözüm, yapay zekanın modeller oluşturmak için Doğal Dil İşleme'yi (NLP) kullanabilmesi için özellikleri açıkça düzenlemektir. Örneğin, Davranış Odaklı Geliştirme (BDD), Gherkin dilini kullanarak sistem davranışını Verilen-Ne-O Zaman biçiminde açıklar. BDD, Agile ekipler tarafından ortak bir dil kullanarak geliştiriciler, test uzmanları ve iş insanları arasındaki iletişimi iyileştirmek için kullanılır.

Bu makine öğrenme yöntemleri, test agentsının başlangıç ve istenen son durumları ana hatlarıyla belirten Gherkin dilindeki özellikleri aldığı hedef odaklı testlere uygulanmıştır. agent, uygulama modeli grafiklerini ve üst düzey test planlarından gelen talimatları kullanarak sistemle etkileşime girer.

Bu bağlamdaki farklı makine öğrenme teknikleri şunlardır:

- Derin Öğrenme ve Yapay Sinir Ağları (ANN)
- Karar Ağacı Öğrenmesi
- Güçlendirme Öğrenmesi
- Bayes Ağları

Teknik	Açıklama	Yarar
Üretim Kullanımından Öğrenme	Testleri iyileştirmek için gerçek kullanım verilerinden faydalanma	Gerçek kullanıma dayalı olarak en kritik testleri belirleme
Test Senaryosu Önceliklendirme	Kod karmaşıklığını, geçmiş sorunları ve kullanıcı davranışlarını analiz ederek testleri önceliklendirme	Ana hataları erken yakalama
Öngörücü Mutasyon Testi	AI kullanarak önceki testlere dayanarak hata içerebilecek alanları tahmin etme	Yoğun test ihtiyacını azaltma ve zaman kazandırma

Tablo 2: Öğrenmeye Dayalı Teknikler ve Faydaları

Makine öğrenmesinin daha karmaşık bir parçası olan derin öğrenme, beynimizin sinir ağlarından esinlenen matematiksel modelleri içerir. Bireysel 'nöronlardan' oluşan bu ağlar, bilgileri kendi başlarına işleyebilir ancak bağlandığında çok güçlü hale gelir.

Karar ağacı öğrenmesini kullanarak kullanıcı arayüzü öğelerini sınıflandırmak için adımlar şunlardır:

1. Verileri hazırlama ve işleme,
2. Modellenecek özellikleri seçme,
3. Modelin, verilerin farklı bölümlerinden birden fazla ağaç oluşturarak aşırı uyumu önleyen karar ağaçları ve rastgele ormanlar gibi teknikler kullanılarak eğitilmesi. (Ho, 1995)

Q-öğrenimi de dahil olmak üzere takviyeli öğrenme, uygulamaları keşfetmek ve bunlarla etkileşim kurmak için tercih edilen bir yöntemdir. Bu teknik, bir algoritmanın deneme yanılma yoluyla ortamından öğrenmesini, doğru eylemler için ödülleri ve yanlış eylemler için cezalar almasını içerir. (Richard & Andrew, 2018) Zamanla, makine açık talimatlar olmadan en uygun eylemleri öğrenir.

Q-öğrenimi, yapay zekanın önceden oluşturulmuş bir model olmadan geçmiş deneyimlerden en iyi eylemi belirlediği bir takviyeli öğrenme türüdür. Kararlar, zaman içinde ödülleri en üst düzeye çıkarmayı amaçlayan bir formüle dayanır, ancak sistemin bazen yeni olasılıkları keşfetmek için rastgele eylemler seçmesi de gerekir. (Siraj, 2017) Bu yöntem, çeşitli oyun agentlarını otomatikleştirmek için kullanılmıştır.

Q-öğrenmesinin avantajları arasında genel kullanıma uyarlanabilirlik, her zaman yeni olan yeniden kullanılabilir ve etkili testler oluşturma yer alır.

Bayes ağları, bir olayın diğer olaylara göre gerçekleşme olasılığıyla ilgili olan koşullu olasılığa dayanır. Daniel Rodriguez, Javier Dolado ve Javier Tuya 2015 yılında yazılım testinde Bayes kavramlarını incelediler. Yazılım testinde Bayes yöntemlerini kullanan 40 farklı çalışmayı incelediler. Araştırmaların çoğu test yürütme veya sonuçları tahmin etmeye odaklandı, bazıları yazılım test çabası tahminlerini ve test üretimini ele aldı. Ağlar, Markov modelleri ve Naïve Bayes sınıflandırıcıları gibi Bayes yöntemleri tahmin ve optimizasyon için kullanılır. (Rodriguez, et al., 2015) Naïve Bayes sınıflandırıcısı, bu teoremin makine öğrenmesinde dikkate değer ve etkili bir uygulamasıdır. (Khyati, 2020)

3.5.6. Yapay Zeka Testinin Riskleri

Bu sistemlerin karmaşık yapısı nedeniyle AI testi zor olabilir. 3.3.4. bölümde daha önce belirtilen tüm zorluklar göz önünde bulundurularak testler akılda tutularak yapılmalıdır. Karmaşıklığı ele almak için bir plana ihtiyaç vardır. (Tariq M, 2018)

ISTQB (ISTQB, 2018) tarafından özetlendiği gibi, AI testi için geleneksel test otomasyonunu kullanmanın içerdiği riskler aşağıda tartışılmaktadır.

AI araçlarına yönelik yüksek beklentiler, özellikle mevcut AI vızıltısıyla birlikte hayal kırıklığına yol açabilir. AI yetenekleri iddia eden bazı araçlar, istenen ihtiyaçları karşılamayabilir.

AI test araçları için ilk kurulum maliyeti, AI, QA stratejistleri, veri bilimcileri ve AI test uzmanlarının eğitilmesi ve işe alınması ihtiyacı dahil olmak üzere hafife alınabilir.

AI test araçlarından kalıcı değer elde etmek için gereken devam eden çabayı, test süreçlerini güncellemeyi ve araç kullanımını iyileştirmeyi de içeren, hafife almak da yaygındır. Test otomasyon uzmanları bu risklerin farkındadır ve AI testinin etkinliği konusunda biraz şüphecidir.

AI araçları tarafından oluşturulan test varlıklarının bakımı, beklenenden daha fazla çalışma gerektirebilir. Yapay zeka testi, geleneksel TA'ya kıyasla bu bakımı azaltmayı hedefler.

Yapay zeka araçlarına aşırı güvenme, bunların test tasarımı veya yürütmesini değiştirmesini bekleme veya manuel testin daha uygun olacağı durumlarda otomatik test kullanma riski vardır.

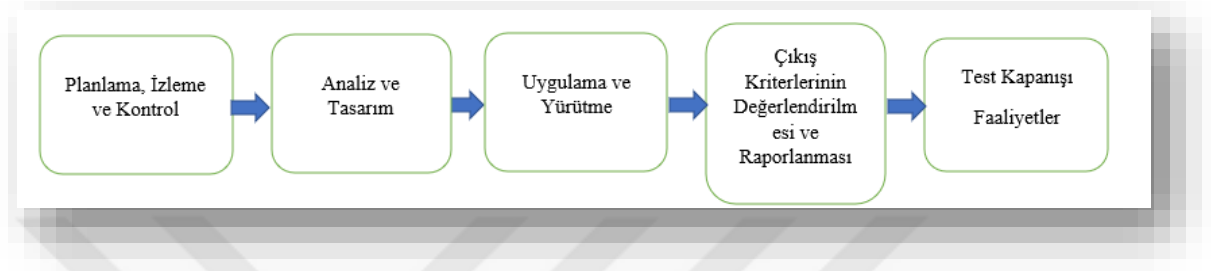
Gereksinimler, yapılandırma ve hata yönetimi gibi diğer kritik araçlarla birlikte çalışabilirlik sorunları göz ardı edilebilir. Yapay zeka testinin bu araçlardan gelen verileri nasıl kullanabileceğini ve bunlarla nasıl entegre olabileceğini düşünmek önemlidir.

Araç satıcısının işi durdurması, aracı durdurması veya yetersiz destek sağlaması riski vardır. Güçlü bir topluluğa sahip açık kaynaklı araçlar seçmek, durdurulma olasılıkları daha düşük olduğu ve geniş bir topluluk desteği sunduğu için bu riski azaltabilir.

Son olarak, yapay zeka test araçlarını Agile veya DevOps uygulamalarıyla entegre etmek bu riskleri yönetmeye yardımcı olabilir. Bu yaklaşım maliyetleri düşük tutabilir ve yeni yapay zeka test yöntemlerine hızlı bir şekilde adapte olmayı sağlayabilir.

3.6. Test Sürecinde Makine Öğrenmesi Yaklaşımları

Bu bölüm, yazılım test sürecinin çeşitli aşamalarını desteklemek için farklı yapay zeka tekniklerinin nasıl kullanılabileceğini açıklamaktadır. Metin, Uluslararası Yazılım Testi Yeterlilikler Kurulu (ISTQB, 2018) tarafından tanımlanan standart bir test sürecine atıfta bulunmaktadır. Bu süreç testleri planlama, izleme, analiz etme, tasarlama, oluşturma, çalıştırma ve bitirme gibi faaliyetleri içerir.



Şekil 8 - Test süreci

(Şekil 8), sonraki bölümlerde daha ayrıntılı olarak tartışacağımız bu sürecin farklı aşamalarını göstermektedir.

ISTQB, test sürecini aşağıdakileri içeren bir dizi bağlantılı aktivite olarak tanımlar:

- **Test Planlama:** Bu, neyin test edilmesi gerektiğini, testin nasıl yapılacağını ve bunu kimin yapacağını bulmayı içerir.
- **Test İzleme ve Kontrol:** Bu, testin ilerleyişini takip etmeyi, her şeyin sorunsuz çalıştığından emin olmayı ve gerektiğinde ayarlamalar yapmayı içerir.
- **Test Analizi:** Bu, herhangi bir hata veya sorunu belirlemek için test sonuçlarının incelenmesini içerir.
- **Test Tasarımı:** Yazılımın nasıl test edileceğine dair bir plan oluşturulmasını içerir.
- **Test Uygulaması:** Bu, test komut dosyalarının yazılması gibi gerçek testlerin oluşturulmasını içerir.
- **Testin Yürütülmesi:** Bu, testlerin yürütülmesini ve sonuçların kaydedilmesini içerir.
- **Testin Tamamlanması:** Bu, test sürecinin tamamlanmasını, sonuçların belgelenmesini ve bulunan hataların rapor edilmesini içerir.

Test uzmanları, bu sürecin her aşamasında AI tekniklerini kullanarak çalışmalarının verimliliğini ve etkinliğini artırabilir. Aşağıdaki bölümlerde yapay zekanın bu aşamaların her birinde nasıl kullanılabileceği tartışılacaktır.

3.6.1. Planlama, İzleme ve Kontrol

Testin ilk aşaması planlamadır. Bu, hangi kaynaklara ihtiyaç duyulacağı ve testi kimin yapacağı gibi, yazılımı doğru bir şekilde test etmek için yapılması gereken her şeyin anlaşılmasını içerir. Bu plan, proje için seçilen genel test stratejisine dayanmaktadır.

Testin bir diğer önemli kısmı izlemedir. Bu, testin nasıl gittiğini takip etmek ve her şeyin yolunda gittiğinden emin olmak anlamına gelir. Ekibin ayrıca test sürecini de kontrol edebilmesi gerekiyor; bu, herhangi bir sorun olması veya bir şeylerin değişmesi gerekiyorsa ayarlamalar yapılması anlamına geliyor.

Testin bu planlama, izleme ve kontrol aşamasında yapay zekanın kullanılabileceği bazı yollar şunlardır:

- **İnsanların Yazılımı Nasıl Kullandığından Öğrenmek:** AI, insanların yazılımı gerçek dünyadaki durumlarda gerçekte nasıl kullandıklarına ilişkin verileri analiz edebilir. Bu bilgiler, hata oluşma olasılığı daha yüksek olan ve daha fazla teste ihtiyaç duyan alanların belirlenmesine yardımcı olmak için kullanılabilir.
- **Testlere Öncelik Verme:** AI, test uzmanlarının hangi testlerin ilk önce çalıştırılmasının en önemli olduğuna karar vermesine yardımcı olabilir. Bu, test edilen özelliğin karmaşıklığı veya bir hata bulma olasılığı gibi faktörler dikkate alınarak yapılabilir.
- **Test Kalitesinin Kontrol Edilmesi:** AI, test paketinin kalitesini otomatik olarak kontrol etmek için kullanılabilir. Bu, eksik testlerin veya çok etkili olmayan testlerin belirlenmesine yardımcı olabilir.

Test uzmanları, bu yapay zeka tekniklerini kullanarak daha iyi planlar yapabilir, ilerlemelerini daha etkili bir şekilde izleyebilir ve başarılı bir test süreci sağlamak için gereken ayarlamaları yapabilir.

3.6.1.1. Üretim Kullanımından Öğrenmek

Sistemlerimizi gerçek kullanıcıların eylemlerinden ve deneyimlerinden öğrenerek öğrenci haline gelecek şekilde donattığınızı hayal edin. Bu gelişmiş izleme, yalnızca hataları yakalamanın ötesine geçer. Bireysel oturumlara dayalı ayrıntılı modeller oluşturarak her kullanıcı etkileşimini yakalayabiliriz. Neyse ki Splunk gibi bu kadar büyük miktarda veriyi işleyecek ve analiz edecek güçlü araçlar var (Splunk, 2024).

Yazılım geliştirme alanındaki uzmanlar, gerçek kullanıcı davranışını ve üretim istatistiklerini analiz etmeyi gelecekteki gelişmeler için bir hazine olarak görüyor (Eric, 2019). İnsanların ürünle nasıl etkileşim kurduğunu ve hataları neyin tetiklediğini inceleyerek test uygulamalarımızı önemli ölçüde geliştirebiliriz. Bu altın madeni bilgi, testlere öncelik verme konusunda bize rehberlik edebilir ve hatta tamamen yeni testlerin oluşturulmasını otomatikleştirebilir. Faydaları temel işlevsellik testlerinin çok ötesine uzanır. Performans, güvenlik ve erişilebilirlik gibi işlevsel olmayan yönlerle ilişkin içgörüler elde edebiliriz (Crispin, 2019). Araştırmalar, ürün kalitesini sürekli olarak izlemek, olası kusurları tahmin etmek ve sorunlara neden olabilecek belirli yazılım modüllerini belirlemek için en ileri teknikleri bile araştırıyor (Rodriguez, et al., 2015). Bu yaklaşımı benimseyerek gerçek dünyayı dev bir test laboratuvarı olarak kullanabilir, olağanüstü bir kullanıcı deneyimi için yazılımımızı sürekli öğrenip geliştirebiliriz.

3.6.1.2. Test Senaryosu Önceliklendirmesi

Her şeyi test etmek bir hayal olabilir ancak kaynaklar genellikle sınırlıdır. Test senaryosunun önceliklendirilmesi, test çabalarımızı öncelikle en önemli alanlara odaklamamıza yardımcı olur. Bu, bazılarının tamamen test edilmeden bırakılabileceği testleri seçmek veya en aza indirmekten farklıdır. Önceliklendirme, testleri aynı anda birden fazla makinede çalıştırırken bile her şey kapsanana kadar testi sürdürmemize olanak tanır (Remo L, 2018).

Test senaryolarını önceliklendirmenin farklı yolları vardır ve en iyi yöntem, spesifik duruma bağlıdır. Bazı teknikler maliyetleri dikkate almazken bazıları dikkate alır. Araştırmacılar tek bir "herkese uyan tek beden" yaklaşımının olmadığını bulmuşlardır (Chen et al., 2018).

Gelecek vaat eden yaklaşımlardan biri, Tahmine Dayalı Test Önceliklendirmesi adı verilen makine öğrenimini kullanıyor. Bu yöntem, bir testin sistemin ne kadarını kapsadığı, çalıştırılmasının ne kadar sürdüğü ve dakikada ne kadar kapsam sağladığı gibi faktörleri analiz eder. Bu verilere dayanarak, belirli bir uygulama için en etkili önceliklendirme tekniğini tahmin ederek, rastgele seçime kıyasla sonuçları önemli ölçüde iyileştirir.

Bir diğer heyecan verici alan da takviyeli öğrenme yoluyla yapay zekanın kullanılmasıdır. Araştırmacılar, sürekli bir entegrasyon ortamında test senaryolarının otomatik olarak önceliklendirilmesi ve seçilmesi için bu tekniği araştırdılar (Spieker, et al., 2017).

Testleri stratejik olarak önceliklendirerek test kaynaklarımızdan en iyi şekilde yararlanabilir ve mümkün olan en iyi yazılım kalitesini sağlayabiliriz.

3.6.1.3. Tahmine Dayalı Mutasyon Testi

Düzenli testler, yazılımımızın doğru çalıştığından emin olmanıza yardımcı olur. Peki testlerimizin iyi olup olmadığını nasıl bileceğiz? Mutasyon testi, test paketlerimizin kalitesini değerlendirmemize yardımcı olan bir tekniktir. Yazılımın "mutantlar" adı verilen biraz değiştirilmiş versiyonlarını oluşturarak çalışır. Daha sonra testlerimizi bu mutantlar üzerinde yapıyoruz ve hataları yakalayıp yakalamadıklarını görüyoruz.

Mutasyon testi değerli bir araç olmasına rağmen yavaş olabilir çünkü her mutantın ayrı ayrı test edilmesi gerekir. Araştırmacılar, Tahmine Dayalı Mutasyon Testi adı verilen yeni bir yaklaşım geliştirdiler. Bu yöntem, tüm mutantları çalıştırmadan mutasyon testinin sonuçlarını tahmin etmeye çalışır. İlk araştırmalar bu yöntemin umut vaat ettiğini gösteriyor (Zhang, et al., 2016).

3.6.1.4 Kaliteli Modeller

Bayes istatistikleri adı verilen ve yazılım kalitesine yönelik modeller oluşturmak için kullanılabilecek güçlü bir istatistiksel yöntem vardır. Bu modeller, aktiviteye dayalı kalite modeli gibi çeşitli görevlerde faydalıdır (Rodriguez et al., 2015):

- Bir sistemin ne kadar güvenilir olduğunu takip etmek

- Test için ne kadar çaba ve zamana ihtiyaç duyulduğunun belirlenmesi
- Yapılacak testlere ilişkin bir plan oluşturmak

3.6.2. Analiz ve Tasarım

Test analizi: Bu, sistem hakkındaki arka plan bilgilerini ("test esası") inceleyerek neyin test edilmesi gerektiğini anladığımız ilk adımdır. Amaç, test hedeflerimize ulaşmamıza yardımcı olacak farklı test türlerini belirlemektir. AI testinde bu analizi yapmanın bir yolu, örneğin UI ne tür öğelerin oluşturduğunu bularak sistemin kendisini keşfetmektir.

Test tasarımı: Neyi test edeceğimizi öğrendikten sonra gerçek test senaryolarını tasarlarız. Bu, bize en kapsamlı testleri ("test kapsamı" olarak adlandırılır) sağlayacak tekniklerin seçilmesini içerir. Sonunda, ilk adımda belirlediğimiz her şeyi (test koşulları) kapsayan ayrıntılı test senaryoları oluşturuyoruz. Bu test senaryoları, test edilen özelliğin veya sistemin karmaşıklığına bağlı olarak daha genel terimlerle (yüksek-düzey) veya daha spesifik ayrıntılarla (düşük-düzey) yazılabilir.

3.6.2.1. Test Akışı Tasarımı

AI-destekli testlerimizin akışını tasarlarken iki ana yaklaşımdan yararlanabiliriz: bildirimsel test ve doğal dil işleme.

1. Bildirime Dayalı Test: Yapay Zekaya Ne Yapacağını Söylemek

Bildirime dayalı testler, bir testin amaçlarını ve adımlarını, en ince ayrıntılara takılıp kalmadan, açıkça tanımlamamıza olanak tanır.

Temel olarak AI test agentsine neyi başarmasını istediğimizi ve bunu nasıl yapması gerektiğini söyleriz. Bu, düz İngilizce veya test edilen yazılıma özel bir dilde yapılabilir.

Bu yaklaşımın harika bir örneği, herkesin katkıda bulunabileceği ve testlerin tanımlanma biçimini geliştirebileceği ortak bir çalışma olan "Soyut Niyet Testi sözdizimi"dir (Jason A, et al., 2018).

2. Doğal Dil İşleme: İnsan Test Kullanıcılarını Taklit Etmek

Bir başka heyecan verici yaklaşım, NLP'nin Uzun Kısa Süreli Bellek Tekrarlayan Sinir Ağları (RNN'ler) adı verilen belirli bir tür sinir ağıyla birleştirilmesini içerir. Bu güçlü teknik, Ultimate Software tarafından AGENT prototipinde kullanıldı. Burada AI agentsı, insan testçilerle birlikte çalışarak onların eylemlerinden öğreniyor.

"Ad" metin alanına sahip bir formu test ettiğinizi düşünün. Bir insan test uzmanı önce alanı gözlemleyebilir, ardından geçerli ve geçersiz verileri girmeyi deneyebilir. NLP sistemi bu eylemleri gözlemleyerek test sürecindeki sonraki adımları tahmin etmeye çalışacaktır. Örneğin, "Ad" alanını gözlemledikten sonra, bir sonraki adımın bu alana bir şey girmeyi "denemek" olduğunu tahmin edebilir. Sistem daha sonra test cihazının geçerli veya geçersiz verileri deneyebileceğini tahmin ederek testin belirli bir senaryoya odaklanmasını sağlayabilir. Son olarak sistem, testin başarılı mı yoksa başarısız mı olduğunu doğrulamak için bir hata mesajının gözlemlenip gözlemlenmeyeceğini tahmin edebildi (Tariq M, 2018) (Santiago, et al., 2018).

Bu yenilikçi tekniklerden yararlanarak yapay zekanın gücüyle yönlendirilen daha akıcı ve etkili bir test süreci oluşturabiliriz.

3.6.3. Uygulama ve Yürütme

Testlerimizi analiz ve tasarım yoluyla titizlikle planladıktan sonra sıra onları hayata geçirmeye geldi! Bu iki temel aşamayı içerir: uygulama ve yürütme.

1. Planın Hayata Geçirilmesi: Uygulama

Testin uygulanması tamamen iyi tanımlanmış test planımızı uygulanabilir adımlara dönüştürmekle ilgilidir. Bu içerir:

- Otomatik testlerin kendilerinin oluşturulması (varsa)
- Verimli yürütme için testleri mantıksal bir sırayla düzenlemek
- Test verilerinin sonuçlandırılması ve test ortamının hazır olmasının sağlanması
- Açık bir yürütme programı oluşturma

İşte yapay zekanın parladığı yer: AI-destekli test araçlarının çoğu "otomatik-iyileştirme" ve "kendi kendini koruma" özelliklerine sahiptir. Bu, araç işlevlerine ilişkin araştırmaların da gösterdiği gibi, yazılımın kullanıcı arayüzündeki değişikliklere dayalı

olarak testleri otomatik olarak düzeltebilecekleri veya güncelleyebilecekleri anlamına gelir.

2. Testlerin Çalıştırılması: Yürütme

Testin yürütülmesi, kauçuğun yolla buluştuğu yerdir. Bu aşamada testleri uygulama aşamasında oluşturulan plana göre yürütüyoruz. İlginçtir ki bu süreç dinamik olabilir. Kapsamlı ek kurulum gerektirmeden, yürütme sırasında keşfedilen bulgulara dayalı olarak yeni testler eklenebilir.

Ancak test yürütme, "test oracle sorunu" olarak bilinen bir zorluğu da beraberinde getirir. Bu, bir test sonucunun doğru olup olmadığını kesin olarak bilmenin zorluğunu ifade eder.

3.6.3.1. Hizmet Sanallaştırma

Ayrı hizmetlerle uygulama oluşturmak zor olabilir. Geliştiriciler ve test uzmanları çeşitli sorunlarla karşılaşabilir, örneğin:

- Henüz tamamlanmamış veya hatalı yanıtlar veren hizmetler
- Sınırlı kapasite veya kullanılamayan hizmetler
- Hizmetlere erişim zorluğu veya farklı yapılandırmaları kullanma kısıtlamaları

Bu hizmetler durumsuz veya durumlu olabilir. Belirli bir şehir için hava durumu tahmini gibi durumsuz hizmetler, geçmiş etkileşimleri hatırlamaya gerek kalmadan basit bir yanıt sağlar. Öte yandan durumlu hizmetler, alışveriş sepetinin ne eklediğinizi hatırlaması gibi geçmiş etkileşimleri takip eder. Durumlu hizmetleri simüle etmek özellikle zor olabilir, ancak makine öğrenimi umut verici çözümler sunar.

Makine öğrenimi bu süreçte üç ana adımı içeren önemli bir rol oynar: yakalama, modelleme ve simülasyon.

1. Yakalama: Bu, gerçek hizmet arasındaki istekleri ve yanıtları kaydetmeyi (kullanılabilir ve güvenilir olduğunda) veya makine öğrenimi algoritmalarının öğrenmesi için bu etkileşimleri manuel olarak oluşturmayı içerir.

2. Modelleme: Yakalanan etkileşimler, hizmetin davranışının bir modelini oluşturmak için kullanılır.

3. Simülasyon: Sanal bir hizmet dağıtılır ve gerçek hizmete güvenmek yerine, aldığı isteklere göre yanıtlar üretir ve gerçek hizmetin davranışını taklit eder.

Araştırmacılar bu modelleri oluşturmak için iki yaklaşımı karşılaştırdılar:

- **Sınıflandırma Tabanlı Sanallaştırma (CBV):** Bu yöntem, RIPPER adı verilen belirli bir algoritma kullanır ve diğer yaklaşımdan daha hızlı eğitilir.
- **Sıradan Sıraya Tabanlı Sanallaştırma (SSBV):** Bu yöntem, Uzun Kısa-Süreli Bellek Ağları adı verilen bir tür sinir ağından yararlanır ve CBV'den daha yüksek doğruluk sunar. Ancak, bir SSB V modelini eğitmek daha uzun sürer ve GPU'lar gibi güçlü donanımlar gerektirir.

CBV eğitilmesi daha hızlı olsa da, SSBV üstün doğruluk sunar. Zaman ve güçlü donanım mevcutsa, SSBV daha iyi bir seçim olabilir. Bu yaklaşım, geliştiricilerin ve test uzmanlarının gerçek hizmet kullanılmadığında veya güvenilir olmadığında bile sorunsuz bir şekilde çalışmasını sağlar.

3.6.3.2 Test Verisi Oluşturma

Bir test platformu olan Mabl, test verisi oluşturma için makine öğrenimiyle heyecan verici olasılıkları araştırıyor. Lisa Crispin'in Mabl web sitesindeki blog yazısına göre, bir yapay zekayı bazı akıllıca numaralar yapması için eğitmeyi düşünüyorlar (Crispin, 2019). İşte fikir: AI, gerçek-dünya verilerinin (üretim verilerinin) büyük bir örneğini analiz edecek, hassas bilgileri korumak için gizleyecek ve ardından özellikle test amaçları için yepyeni bir test verisi seti oluşturacak.

Bu neden gerekli? Çünkü testlerde gerçek üretim verilerini kullanmak, olası varyasyonların çokluğu nedeniyle zor olabilir. Makine öğrenimiyle bu yenilikçi yaklaşım, oyunun kurallarını değiştirebilir.

Bir diğer umut verici yol ise test verisi oluşturma için Bayes istatistikleri adı verilen istatistiksel bir yöntem kullanmaktır (Rodriguez, et al., 2015). AI ve makine öğrenimindeki bu ilerlemeler, test verisi oluşturma sürecini kolaylaştırmak ve iyileştirmek için muazzam bir potansiyele sahiptir.

3.6.3.3. Diferansiyel Test

Farklı test, bir uygulamanın çeşitli sürümleri arasındaki farkları belirlememize yardımcı olan güçlü bir tekniktir (Tariq M, et al., 2019). İşleyişi şöyledir: hem eski hem de yeni sürümlerde aynı testleri çalıştırırız ve sonuçları karşılaştırırız. Bu şekilde, yalnızca işlevsellikteki değişiklikleri değil, aynı zamanda verimlilikteki değişiklikleri veya belirli görevleri tamamlamanın ne kadar sürdüğünü de belirleyebiliriz. Bu verileri analiz ederek, gelişen uygulamayı daha iyi anlamak için test modellerimizi güncelleyebiliriz.

3.6.3.4. Görsel Test

Görsel test, uygulamamızın olması gerektiği gibi görünmesini ve hissettirmesini sağlamakla ilgilidir. Uygulamayı bizim gibi "görebilen" özel araçlar kullanmayı hayal edin (Tariq M, et al., 2019). Bu araçlar, her şeyin amaçlandığı gibi görünüp görünmediğini kontrol etmek için görüntü tanıma ve ekran karşılaştırmaları kullanır.

Harika bir örnek, bir test platformu olan Mabl'dır. Mabl, bir kullanıcının uygulamada gezinirken kaydedebilir ve ziyaret edilen her sayfanın görsel bir modelini oluşturabilir (Eric, 2019). Genellikle sayfa başına en az on ekran görüntüsü alırlar ve sık görsel değişiklik yapılan sayfalar için daha da fazla alabilirler. Bu ekran görüntülerini zaman içinde karşılaştırarak Mabl, beklenmeyen görsel değişiklikleri belirleyebilir ve bunların tam olarak nerede meydana geldiğini belirleyebilir.

Bu tür testler yalnızca estetik açıdan değil, işlevsellik ve kullanılabilirlik açısından da değerlidir. Görsel bir değişiklik bir şeyin çalışma şeklini bozarsa veya kullanımını zorlaştırırsa, görsel test bu sorunları erken yakalamamıza yardımcı olabilir.

3.6.3.5. Testin Uygulanması ve Yürütülmesi için Bayes Yaklaşımları

Bayes teoremi, test uygulama ve yürütmenin çeşitli alanlarında kullanılan yararlı bir yöntemdir. İşte nasıl kullanıldığına dair bir döküm:

- **Test Tasarımı ve Veri:** Bu yöntem, testlerin ve bu testleri çalıştırmak için gereken verilerin oluşturulmasına yardımcı olur (Rodriguez, et al., 2015).

- **Doğru Testleri Seçme:** Hangi testlerin önce çalıştırılacağına karar vermeye yardımcı olabilir ve en önemlilerine öncelik verebilir (Rodriguez, et al., 2015). Bu, özellikle yazılımda değişiklikler yapıldıktan sonra testlerin yeniden çalıştırılmasını içeren regresyon testinde yararlı olabilir.
- **Test Yürütme İyileştirmeleri:** Bayes teoremi, testlerin nasıl çalıştırılacağını iyileştirmek için de kullanılabilir. Örneğin, grafiksel kullanıcı arayüzlerinin otomatik testi sırasında ortaya çıkan sorunları sınıflandırmaya yardımcı olabilir (Martinez, et al., 2014). Ek olarak, sorunlara neden olmadan önce bile yazılımdaki hataları ve kusurları tahmin etmek için kullanılabilir (Rodriguez, et al., 2015). Bu, yazılımın arızalı olabilecek belirli bölümlerini belirlemeye ve ayrıca bir testin güvenilir olmayan sonuçlar üretip üretmeyeceğini tahmin etmeye yardımcı olabilir (Tariq M, 2018).

3.6.3.6. Test Oracle

AI sistemlerini test etmek benzersiz bir zorluk sunar: Doğru cevabın ne olduğunu nasıl bileceğiz? Geleneksel yazılım testi, çıktıyı önceden belirlenmiş, beklenen bir sonuçla karşılaştırmaya dayanır. Ancak, AI sistemleri doğaları gereği karmaşık kalıplara ve olasılıklara göre öğrenir ve uyum sağlar. Bu, her durum için tek bir "doğru" cevap tanımlamayı zorlaştırır.

Şu anda, AI sistemlerini test etmek büyük ölçüde insan uzmanlığına bağlıdır. Test uzmanları, AI'nın çıktılarını analiz etmek ve bunların makul görünüp görünmediğini ve sistemin amaçlanan amacıyla uyumlu olup olmadığını belirlemek için eleştirel düşünme becerilerini kullanırlar. Bu yaklaşım değerli olsa da zaman alıcı olabilir ve bireysel önyargılarla sınırlı olabilir.

Araştırmacılar, insan yargısına olan bağımlılığı azaltmayı ve verimliliği artırmayı hedefleyerek test oracle sürecini otomatikleştirmenin yollarını aktif olarak araştırıyorlar. İşte bazı umut verici tekniklere bir bakış:

- **Deneyimden Öğrenme (Braga et al., 2018):** Geçmiş performansını izleyen bir öğretmen tarafından sürekli izlenen bir öğrenciyi hayal edin. Bu yaklaşım benzerdir. Araştırmacılar, AI sisteminin geçmişte çeşitli girdilerle nasıl

davrandığına dair veri toplamayı öneriyorlar. Bu tarihsel veriler daha sonra yeni durumlar için beklenen çıktıyı tahmin etmek üzere bir makine öğrenimi modelini eğitmek için kullanılır. Esasen, model gelecektekiler hakkında eğitimli tahminlerde bulunmak için AI sisteminin geçmiş kararlarından öğrenir.

- **Stratejik Test Gruplandırması (Roper, 2019):**Bu yöntem, test vakalarını sonuçlarına göre gruplandırarak verimliliğe odaklanır. Sürekli olarak benzer çıktılar, özellikle başarısızlıklar üreten testlerin aynı temel sorundan etkilenmesi muhtemeldir. Bu kümeleri manuel inceleme için önceliklendirerek, test uzmanları sorunları daha hızlı belirleyebilir. Sınıflandırılmamış bir giysi yığınının hayal edin. Bunları türe (gömlekler, pantolonlar) göre gruplamak, aradığınızı bulmanızı kolaylaştırır. Bu yöntem aynı mantığı test vakalarına uygular.
- **Hatalardaki Desenleri Belirleme (Dwarakanath et al., 2018):** Tüm yollar Roma'ya çıkmaz, ancak biri bunun yerine Tokyo'ya giderse, muhtemelen bir yerde yanlış bir dönüş vardır. Bu teknik benzer şekilde çalışır. Araştırmacılar, AI sistemi için farklı girdiler ve bunlara karşılık gelen çıktılar arasında beklenen ilişkileri kurmaya odaklanır. Yapay zekanın çıktısı bu yerleşik kalıplardan önemli ölçüde saparsa, bu sistem kodunda bir hata olduğunu gösterebilir. Bu, tek yönlü bir sokakta yanlış yöne giden bir araba fark etmek gibidir - bir şeylerin doğru olmadığına dair açık bir işarettir.

Bunlar, AI testinde test oracle sorununu ele almak için devam eden araştırma çabalarının sadece birkaç örneğidir. Bu alan gelişmeye devam ettikçe, AI-destekli sistemlerin daha güvenilir ve verimli bir şekilde test edilmesinin önünü açarak daha da yenilikçi çözümlerin ortaya çıkmasını bekleyebiliriz.

4. BULGULAR

Bu bölümde araştırmanın bulgularını inceleyeceğiz. Test yönetimini, statik testi, test tasarımını ve yürütülmesini, performans analizini ve güvenlik testini destekleyen araçları analiz edeceğiz. Ayrıca test süreçlerini iyileştirmek için kullanılan ve bu süreçlerin etkinliğinin ve verimliliğinin artırılmasına katkıda bulunan genel AI araçlarını ve çerçevelerini tartışacağız.

4.1. Yazılım Testi için Yapay Zeka Araçları

İlk olarak, bu araçların genellikle nasıl kategorize edildiğini açıklayacağız. Ardından, yapay zekanın bu araçların tasarımını ve işlevselliğini etkileyerek işleri nasıl değiştirdiğini ve sonuçta bunları yazılımı test etmek için kullanma şeklimizi nasıl değiştirdiğini keşfedeceğiz. Özetlemek için, temel işlevlerinde makine öğreniminden yararlanan test araçlarının gerçek dünyadan bazı örneklerini sergileyeceğiz. Bu araçları sınıflandırma şeklimiz, köklü ISTQB yazılım testi müfredatına dayanmaktadır (ISTQB, 2018).

Araç Kategorisi	Ticari Araçlar	Açık Kaynak Araçlar
Test Yönetimi ve Testware	Testim.io, Perfecto	Selenium, Appium
Statik Test Desteği	SonarQube, Checkmarx	FindBugs, PMD
Test Tasarımı, Uygulama, Yürütme ve Kayıt Desteği	Tricentis Tosca, Ranorex	Katalon Studio, Robot Framework
Performans Ölçümü ve Dinamik Analiz Desteği	LoadRunner, JMeter	Gatling, Locust
Diğer Özelleşmiş Test İhtiyaçları Desteği	Burp Suite (Güvenlik Testi)	OWASP ZAP (Güvenlik Testi)

Tablo 3: Yazılım Testine Yönelik Yapay Zeka Araçları

4.1.1. Test ve Test Yazılımının Yönetimini Destekleyen Araçlar

Doğru test yönetimi araçlarını seçmek zor olabilir. Çoğu araç, testin çeşitli aşamalarını kapsar veya yazılım testinin farklı alanlarında kullanılan diğer araçlarla yakın şekilde çalışabilir.

Bu araçlar, gereksinimleri ve spesifikasyonları yönetmek, test senaryolarını tasarlamak, testleri gereksinimlerle ilişkilendirmek, testleri çalıştırmak ve test sırasında bulunan hataları (kusurları) yönetmek gibi farklı görevlerde yardımcı olabilir.

Yapay zekanın geleceği ve test yönetiminde nasıl kullanılacağı belirsizliğini koruyor. Yapay zekanın ne kadar gelişmiş olacağına bağlı.

Geleneksel olarak ISTQB, test yönetimi araçlarını şu kategorilere ayırır:

- **Test Yönetimi ve Uygulama Yaşam Döngüsü Yönetimi (ALM) Araçları:** Bu araçlar, genel test sürecini ve bunun yazılım geliştirme döngüsüne nasıl uyduğunu ele alır.
- **Gereksinim Yönetimi Araçları:** Bu araçlar, sistemin gereksinimlerinin ve spesifikasyonlarının yönetilmesine yardımcı olur.
- **Kusur Yönetimi Araçları:** Bu araçlar, test sırasında bulunan hataları (kusurları) izler ve yönetir.
- **Konfigürasyon Yönetim Araçları:** Bu araçlar, test edilen yazılımın farklı versiyonlarının yönetilmesine yardımcı olur.
- **Sürekli Entegrasyon Araçları:** Bu araçlar, kod değişikliklerinin sık sık entegre edilmesi ve test edilmesi sürecini otomatikleştirir.

Gelecekte AI, kullanıcıların yazılımla nasıl etkileşim kurduğuna ve performansına bağlı olarak sistem gereksinimleri için iyileştirmeler bile önerebilir. AI ayrıca gereksinimlerin nasıl tanımlandığını etkileyerek iş süreçlerinin yönetilmesine de yardımcı olabilir. Örneğin, Meinke ve Bennaceur tarafından yapılan bazı araştırmalar, mevcut sistemlerden spesifikasyonların otomatik olarak çıkarılması için makine öğreniminin kullanılmasını araştırdı (Meinke ve Bennaceur, 2017).

Şu anda kusur yönetimi araçları, hataların manuel olarak izlenmesine ve düzeltilmesine dayanmaktadır. Ancak AI testi ve geliştirmesindeki ilerlemelerle birlikte bu araçlar, kendi

kendini test eden ve kendi kendini düzelten sistemlerin bir parçası haline gelebilir ve potansiyel olarak kod hatalarını otomatik olarak düzeltebilir.

4.1.2. Statik Testi Destekleyen Araçlar

Statik kod analizi için kullanılan araçlar bile Yapay Zekanın yardımıyla daha akıllı hale geliyor. Aslında DeepCode adlı İsviçreli bir girişim, kod incelemelerini otomatikleştirmek için makine öğrenimini kullanarak Ağustos 2019'da büyük bir sıçrama yaptı. Bu, kendilerine 4 milyon ABD doları sağlayan yatırımcıların dikkatini çekti.

Güzel haberler? AI-destekli bu araç, 30'dan az geliştiricisi olan küçük şirketler, açık kaynaklı projeler ve eğitim amaçlı olanlar için ücretsizdir (Lomas, 2019).

4.1.3. Test Tasarımını, Uygulamayı, Yürütmeyi ve Günlüğe Kaydetmeyi Destekleyen Araçlar

Bu bölüm, planlamadan yürütmeye kadar test sürecinin çeşitli aşamalarında yardımcı olan araçları kapsar.

- **Test Tasarım Araçları:** Bu araçlar açık ve iyi organize edilmiş test senaryoları, prosedürler ve veriler oluşturmayı kolaylaştırır.

Aşağıda test tasarım araçlarına ilişkin bazı özel örnekler verilmiştir:

- **Model Tabanlı Test Araçları:** Bu araçlar, yazılımın nasıl çalışması gerektiğine ilişkin modellere veya simülasyonlara dayalı olarak testlerin tasarlanmasına yardımcı olur.
- **Test Verisi Hazırlama Araçları:** Bu araçlar, test uzmanlarının testlerini yürütmek için ihtiyaç duyduğu verilerin oluşturulmasına yardımcı olur.
- **ATDD (Kabul Testi Odaklı Geliştirme) ve BDD (Davranış Odaklı Geliştirme) Araçları:** Bu araçlar, yazılım için açık kabul kriterleri tanımlamak üzere test uzmanları ve geliştiriciler arasındaki işbirliğini teşvik eder.

- **TDD (Test Odaklı Geliştirme) Araçları:** Bu araçlar, kod yazmadan önce test yazmayı teşvik ederek kodun planlandığı gibi çalışmasını sağlamaya yardımcı olur.
- **Test Yürütme ve Günlüğe Kaydetme Araçları:** Bu araçlar, testlerin yürütülmesine ve sonuçların takip edilmesine yardımcı olur. İşte bazı örnekler:

- **Test Yürütme Araçları:** Bu araçlar test çalıştırma sürecini otomatikleştirir.
- **Kapsama Araçları:** Bu araçlar, yazılımın kodunun veya gereksinimlerinin ne kadarının test edildiğini gösterir.
- **Test Donanımları:** Bu araçlar, testleri çalıştırmak için bir çerçeve sağlar.
- **Birim Testi Çerçeve Araçları:** Bu araçlar, kodun ayrı ayrı bölümlerini test eden birim testlerini çalıştırmak için özel olarak tasarlanmıştır.

AI-destekli birçok test tasarımı ve uygulama aracının aynı zamanda test yürütme ve günlüğe kaydetme işlemlerini de gerçekleştirebildiğini unutmamak önemlidir. Dolayısıyla bu bölüm, test sürecinde sıklıkla birlikte çalıştıkları için bu iki tür aracı birleştirir.

4.1.3.1. Ticari Araçlar

Sonraki bölümlerde otomatik teste yardımcı olmak için AI ve ML kullanan araçların örnekleri gösterilecek. Bazı araçlar eylemleri kaydetme ve oynatma gibi şeyler yapabilir. ML, test edilen yazılımda değişiklikler olduğunda bu araçların bozuk testleri

kendi başlarına düzeltmelerine yardımcı olur. Diğer araçlar, yazılımı kendi başlarına incelemek ve yeni testler yapmak için ML'yi kullanır. Yazılıma bakarak değişiklikleri kontrol etmek de bu araçları kullanmanın popüler bir yoludur. Bu araçlarla ilgili bilgiler, onları üreten şirketlerden geldiği için bilimsel olarak test edilmemiştir. Ancak Mabl, Applitools ve SOFY kullanılarak ne kadar iyi çalıştıklarına dair bazı küçük testler yapıldı.

Mabl, eylemleri kaydeden ve ML'nin yardımıyla kendi testlerini düzeltebilen bir araçtır. ML, yazılım değiştikçe testlerin daha iyi olmasına yardımcı olur ve insanlardan fazla yardıma ihtiyaç duymadan iyi geri bildirimler verir. Şirket, Mabl'ın kodlara başka biriyle bakmak, tüm sistemi test etmek ve keşfetmek için iyi olduğunu söylüyor. Ayrıca, birçok teste ve gerçek kullanıcıların yazılımı nasıl kullandığına dayanarak yazılımdaki kalıpları veya olağandışı şeyleri izleyen kaliteli bir araca da sahiptir. Mabl, yazılımın yaptıklarını not alıyor ve bu notlarla bir model oluşturuyor. Testler modeli güncel tutar ve sonuçlar bu modele göre kontrol edilir. (Hristov, 2019)

Mabl, Google'ın Cloud'unda çalışır ve Google'ın Cloud ML Engine, Dataflow ve Lighthouse gibi ML araçlarını kullanır. (Eric, 2019)

Functionize, Uyarlanabilir Dil İşleme adı verilen bir teknolojiyi kullanarak İngilizce yazılı talimatlardan testler yapmak için ML'yi kullanır. Ayrıca gerçek kullanıcıların yaptıklarını izleyerek yeni testler de yapabiliyor. Yazılım değiştiğinde testleri kendi kendine düzeltebilir ve testlere neler ekleneceği konusunda önerilerde bulunur. (Functionize, 2024)

Bu ürünlerin web sitelerine baktığımızda Mabl'ın sunduğu ürünlerle pek çok ortak noktaya sahip olduklarını görüyoruz.

Appliteools, ekranda gördüğünüz şeyin doğru olduğundan emin olmanızı sağlayan bir araçtır. Birçok test çerçevesi ve programlama diliyle çalışır. Ayrıca farklı web tarayıcıları, ekran boyutları ve mobil uygulamalarla dikey ve yatay görünümde çalışır. (Appliteools, 2024)

Appliteools, ekrandakileri ayıklamak ve bunu bir örnek veya resimle karşılaştırmak için AI ve ML'yi kullanıyor. Çok değişen parçaları göz ardı edebilir, böylece yanlış uyarı vermezler. Test yazmak daha kolaydır çünkü ekranın her bölümünü kontrol etmenize gerek yoktur ve sorunlar görsel değişikliklere göre bulunur. Ayrıca çok değişen testlerde yazılımın bazı kısımlarını güncellemenize gerek kalmadığı için testleri güncel tutmak

daha kolaydır. Ekranda yeni bir deęişiklik olduęunda bunu onaylıyorsunuz. Bir sorun varsa, onu tam orada ekranda vurgulayabilirsiniz. Daha sonra bu sorunları, yazılımı yönetmenize yardımcı olan dięer sistemlere bağlayabilirsiniz.

keysight testleri yapmaya ve optimize etmeye yardımcı olur. Yazılımın bir modeline sahiptir, testler yapar, testlerin yazılımın ne kadarını kapsadığını kontrol eder ve hataları arar. Özdeş İkiz adı verilen bu model, yazılımı durum ve eylemler halinde gösterir. ML, bu modeli ve geçmiş testleri temel alan yeni testler oluşturmaya yardımcı olur. Böylece testlerin nerelerde kaçırıldığını görebiliyor ve o kısımlara yenilerini yapabiliyoruz. Bir hata bulduğunda adımları kaydeder ve insanların hatayı düzeltmesine yardımcı olmak için hatayı tekrarlamak için farklı yollar dener. (keysight, 2024)

TestRigor, AI tarafından desteklenen, keşif amaçlı bir test aracıdır. Hiçbir şey kaydetmeye gerek kalmadan bir web sayfasının bölümlerini bulabilir, yazılımın nasıl çalıştığını anlayabilir ve bu anlayışa göre testler yapabilir. Yazılımın hangi bölümlerinin en çok kullanıldığını bağlı olarak hangi testlerin en önemli olduğunu seçer. (Testrigor, 2024)

Testim ile testleri kaydedebilir veya kodlayabilirsiniz. Test edilen yazılım deęiştğinde testleri otomatik olarak düzelteren özelliklere sahiptir. Sayfaların nasıl görüldüğünü kontrol eder ve farklı veriler, koşullar ve tarayıcılarla test etmenize olanak tanır. Yapay zeka, kaydedilen adımların kullanımını ve testler arasında paylaşılmasını kolaylaştırmaya yardımcı olur. (Testim.io, 2024)

Apptest.ai, AI Bot kullanan mobil uygulamalara yönelik bir keşif test aracıdır. (Apptest.ai, 2024)

Sofy.ai, yazılımın her web sayfasından ve ekranından geçen bir keşif test motoru sunar. Yazılımdaki deęişikliklere göre yeni testler önerebilir ve yapabilir. (Sofy, 2024)

perfecto.io, Selenium adlı bir bilgisayar programıyla ekranda nesneleri hareket ettirerek çalıştırabileceğimiz testler yapan bir araçtır. Yazılımın dinamik bir modelini güncel tutmak için yapay zekayı kullanıyor ve bu da testleri sürdürmek için gereken süreyi kısaltıyor. Testler farklı ayarlarda ve farklı tarayıcılarda paralel olarak çalıştırılabilir. (perfecto.io, 2024)

Diffblue Cover, otomatik olarak birim testleri yapmak için yapay zekayı kullanan Java uygulamalarına yönelik bir araçtır. (Diffblue, 2024)

4.1.3.2. Açık Kaynak Araçları

AGENT, makine öğrenimini kullanan ve yardımcılardan oluşan bir ekip (çoklu-agent) gibi çalışan, web uygulamalarına yönelik bir test aracıdır. Ultimate Software tarafından gelecekteki bir AI test sisteminin bir örneği (prototip) olarak oluşturuldu. AGENT, bir web sayfasının farklı bölümlerini (öğelerini) etiketlemeye yardımcı olan, AGENT-X adlı bir Chrome uzantı aracından gelen eğitim verilerini kullanır. (Santiago, et al., 2018)

AGENT, bir web uygulamasını (AUT) keşfetmek ve insan test uzmanlarının geçmişte yaptıklarına dayalı olarak belirli test adımları (test akışları) oluşturmak için çeşitli akıllı programlar (akıllı agentlar) kullanır. Bu geçmiş eylemler, Uzun Kısa Süreli Bellek (LSTM) adı verilen bir makine öğrenimi türünü kullanan bir model (soyut akış) olarak depolanır. Test sırasında agentlar, bir insan testçinin bundan sonra ne yapacağını tahmin etmeye çalışır. Yaptıkları testler, web uygulamasının bu öğrenilen kalıplara dayanarak olması gerektiği gibi çalışıp çalışmadığını kontrol eder.

Özel bir agent (koordinasyon agent) test sürecini yönetir. Test görevlerini farklı agentlara atamak için bir sistem (mesaj kuyruğu) kullanır. Bu, AGENT'in testleri çalıştırmak, web uygulamasını keşfetmek ve yeni testleri daha hızlı oluşturmak için aynı anda (eş zamanlı olarak) birden fazla agent kullanmasına olanak tanır. (AGENT, 2019)

- AGENT'in kullandığı farklı teknolojiler şunlardır (teknoloji yığını):
 - Python – Ortak bir programlama dili
 - Pandas - Python'da verileri analiz etmek için bir araç
 - NumPy - Python'da bilimsel hesaplama için bir araç
 - Scikit Learn - Python'da veri madenciliği ve analizi için bir araç
 - Keras - Python'da derin öğrenme için bir araç
 - TensorFlow - Makine öğrenimi için bir platform
 - Lark - Doğal dili anlamak için bir araç
 - Celery - Farklı bilgisayarlarda çalıştırılan görevleri yönetmek için bir sistem (dağıtılmış görev kuyruğu)
 - Flower - Görevlerin nasıl çalıştığını görmek için bir araç
 - RabbitMQ - Farklı programlar arasında mesaj göndermek için popüler bir araç (mesaj aracı)

- JSON - Bilgisayarlar arasında veri alışverişi için bir format (hafif veri değişim formatı)
- Selenium WebDriver - Bir web tarayıcısını otomatik olarak kontrol etmek için bir araç (tarayıcı otomasyon aracı)

Tüm bu araçlar, Docker'ı kullanarak küçük hizmetler (mikro hizmetler) olarak birlikte çalışır. AGENT varsayılan olarak bir koordinatör aracı ve bir test aracı çalıştırır, ancak gerekirse daha fazla test aracı kullanacak şekilde ayarlayabiliriz. (Tariq M, 2018)

Appium

- Appium, mobil uygulamaları test etmek için kullanılan başka bir açık kaynak araçtır. Ekrandaki öğeleri tanımlamaya yardımcı olmak için makine öğrenimini kullanır. Bu, test yazmayı kolaylaştırır ve testleri daha güvenilir (daha az kırılgan) hale getirir. Test topluluğu ayrıca daha fazla eğitim verisi ekleyerek, farklı eğitim yöntemleri kullanarak veya öğeler için yeni etiketler ekleyerek aracı geliştirebilir. (Tariq M, et al., 2019)

DrQA

- DrQA, soruların yanıtlarını Wikipedia belgelerinde arayarak yanıtlayabilen bir sistemdir. Wikipedia'daki soruları yanıtlamak için kullanılacak kod, veri ve önceden eğitilmiş modelleri içerir. DrQA özel olarak test için tasarlanmasa da, test edilen yazılımın spesifikasyonlarından ve gereksinimlerinden modeller oluşturularak AI testinin sonuçlarını kontrol eden bir sistem (test oracle) oluşturmak için kullanılabilir. (Tariq M, et al., 2019) (DrQA, 2019)

AppFlow

- AppFlow, farklı web siteleri ve uygulamalarda (etki alanları arası) çalışabilen Android uygulamalarını test etmeye yönelik bir araçtır. Kullanıcı arayüzünün ekranlarını ve öğelerini sınıflandırmak için makine öğrenimini kullanır. (Hu, et al., 2018)

4.1.4. Performans Ölçümünü ve Dinamik Analizi Destekleyen Araçlar

Bu bölüm, performansı ölçmek ve uygulamaları çalışırken analiz etmek (dinamik analiz) için kullanılan farklı araçları kapsar. Bazı ticari performans test araçları halihazırda izleme ve dinamik analiz araçlarını içerse veya bunlarla bağlantı kursa da, bu kategori her işlev için ayrı araçlara odaklanır.

- **Performans test araçları:** Bu araçlar, bir uygulamanın hız ve kararlılık gibi alanlarda ne kadar iyi performans gösterdiğini ölçer.
- **İzleme araçları:** Bu araçlar uygulamanın zaman içindeki performansını takip ederek sorunların belirlenmesine yardımcı olur.
- **Dinamik analiz araçları:** Bu araçlar, sorunları veya iyileştirilecek alanları bulmak için bir uygulamayı çalışırken analiz eder.

Performans Testinde AI

Bazı mevcut test araçları, performans özelliklerini ölçmek için yapay zekayı kullanır. Örneğin daha önce bahsettiğimiz Mabl aracı test çalıştırmanın ne kadar sürdüğünü izliyor. Testin ne kadar sürmesi gerektiğine dair bir model oluşturur ve bu modeli her çalıştırmada günceller. Mabl daha sonra testin beklenenden çok daha uzun mu yoksa kısa mı sürdüğünü size söyleyebilir. (Eric, 2019)

AI test sistemi aynı anda birden fazla test uzmanını (agenti) çalıştırabilir. Bu, bir uygulamanın nasıl performans gösterdiğini görmek için uygulamaya ağır bir yük koymayı içeren yük testine benzer olabilir. Ancak gerçek yük testi için uygulamanızın özel ihtiyaçlarına göre test cihazlarını dikkatli bir şekilde planlamanız ve kontrol etmeniz gerekir. Daha gelişmiş AI sistemlerinde bu gereksinimler, uygulamanın gerçek-dünya koşullarında nasıl kullanıldığına bağlı olarak otomatik olarak ayarlanabilir.

Bir AI test sistemi, test sırasında sistem kaynaklarını izlemek ve testleri yazılı gereksinimlere göre ayarlamak için de kullanılabilir. Bu, aynı araçları hem işlevsel testler (uygulamanın nasıl çalıştığını test etmek) hem de performans testleri için kullanabileceğiniz anlamına gelir. Ancak, birçok tarayıcı işleminin aynı anda çalıştırılması gibi bazı işlevsel test yaklaşımlarının, sistemde çok fazla yük oluşturması nedeniyle yük veya stres testi için uygun olmayabileceğini unutmamak önemlidir.

İzleme ve Dinamik Analizde AI

AI ve makine öğrenimini kullanan bir izleme ve dinamik analiz aracına örnek olarak Splunk verilebilir. Splunk, olağandışı etkinlikleri tespit etmek (anormallik tespiti), gelecekteki olayları tahmin etmek (tahminsel analiz) ve benzer verileri bir arada gruplamak (kümeleme) için ML algoritmalarını kullanır. (Splunk, 2024)

Performans testi için bir AI çözümü, Luo ve diğerleri araştırmasına benzer olabilir. Ekipleri, performans sorunlarını tanımlamak için makine öğrenimini kullanan FOREPOST adında bir araç geliştirdi. Geleneksel performans testlerinde test verileri genellikle uygulamaya rastgele bir sırayla beslenir. Bunun nedeni, verileri analiz etmenin ve uygulamayı nasıl etkileyeceğini tahmin etmenin çok zaman alması olabilir. FOREPOST, her test çalıştırmasında hangi verilerin kullanılacağını seçmek için makine öğrenimini kullanır. Bu, testin performans sorunlarına neden olma olasılığı en yüksek olan verilere odaklanmasını sağlayarak darboğazların (uygulamanın yavaşladığı alanlar) belirlenmesini kolaylaştırır. (Luo, et al., 2016)

FOREPOST aracı, açık kaynaklı performans test aracı Apache JMeter'e AI ve makine öğrenimi özellikleri ekleyerek çalışır. Ancak kullandığı özel profil oluşturma aracı nedeniyle FOREPOST şu anda yalnızca Java uygulamalarıyla çalışmaktadır. (Luo, et al., 2016)

Belirli miktarda veriyi işlemek için en verimli donanım yapılandırmalarını bulmak amacıyla makine öğrenimini kullanan J. Wildstrom ve diğerleri tarafından da benzer bir araştırma yapılmıştır. (Luo, et al., 2016)

4.1.5. Diğer Özel Test İhtiyaçlarını Destekleyen Araçlar

Bu bölümde, Uluslararası Yazılım Testi Yeterlilik Kurulu'nun yönergeleri takip edilerek, çeşitli özel test alanlarında kullanılan araçlar incelenmektedir. İşte bazı önemli örneklerin dökümü:

- **Veri Doğrulama Araçları:** Kitapları organize eden bir kütüphaneci olduğunuzu hayal edin. Bu araçlar titiz asistanlar gibi davranarak verilerin doğruluğunu ve

eksiksizliğini sağlar. Eksik girişleri, tutarsızlıkları veya yerinde olmayan diğer bilgileri kontrol ederler.

- **Veri Taşıma Araçları:** Bu araçları uzman taşıyıcılar olarak düşünün. Verilerin bir formattan veya sistemden diğerine güvenli ve etkili bir şekilde aktarılmasını sağlarlar ve bu süreçte hiçbir şeyin kaybolmayacağını veya bozulmayacağını garanti ederler.
- **Kullanılabilirlik Test Araçları:** Bu araçlar, bir uygulamanın ne kadar kullanıcı dostu olduğunu değerlendirir ve insanların uygulama içinde gezinmesinin ve onunla etkileşim kurmasının ne kadar kolay ve sezgisel olduğuna odaklanır. Kullanılabilirlik testi genellikle, daha sonra inceleyeceğimiz gibi, uygulamaların engelli kişiler tarafından kullanılabilir olmasını sağlayan erişilebilirlik testini de içerir.
- **Erişilebilirlik Test Araçları:** Herkes aynı yeteneklere sahip değildir. Bu araçlar, bir uygulamanın görme engeli, işitme güçlüğü veya diğer kısıtlılıkları olan kişiler tarafından etkili bir şekilde kullanılıp kullanılamayacağını kontrol etmede çok önemli bir rol oynamaktadır. Uygulamaların kapsayıcı olmasını ve erişilebilirlik standartlarını karşılamasını sağlamaya yardımcı olurlar.
- **Yerelleştirme Test Araçları:** Dünya çok çeşitliliğe sahip bir yer! Bu araçlar, bir uygulamanın farklı dillerde ve bölgelerde kusursuz şekilde çalışmasını garanti eder. Kültürel nüanslar, tarih ve saat formatları gibi faktörleri göz önünde bulundururlar ve uygulamanın bilgileri hedef kitleye doğru şekilde göstermesini sağlarlar.
- **Güvenlik Test Araçları:** Güvenlik her şeyden önemlidir. Bu araçlar, bir uygulamanın güvenlik sistemindeki bilgisayar korsanları tarafından kullanılacak güvenlik açıklarını tespit ederek dikkatli korumalar gibi davranır. Geliştiricilerin proaktif olarak zayıf noktaları ele almasına ve uygulamaları potansiyel saldırılara karşı korumasına yardımcı olurlar.

- **Tařmabilirlik Test Araçları:** Farklı ortamlara uyum sağlayabilen bir bukaletun hayal edin. Bu araçlar, bir uygulamanın yalnızca belirli bir bilgisayar sisteminde değil, çeřitli bilgisayar sistemlerinde sorunsuz bir řekilde çalışıp çalışmadığını deęerlendirir. Temel donanım veya işletim sisteminden bağımsız olarak uygulama işlevselliğinin tutarlı kalmasını sağlarlar.

Arařtırmacılar sürekli olarak bu özel testlerin etkinliğini artırmanın yollarını arařtırıyorlar. Gelecek vaat eden alanlardan biri makine öğreniminin, görüntü tanımanın ve doğal dil işlemenin kullanılmasını içerir. Örneğın makine öğrenimi, görevleri otomatikleřtirerek ve engelli kullanıcılar için olası engelleri belirleyerek erişilebilirlik testini geliřtirebilir. Bu tür testler kullanılabilirlik testi kapsamına girer. Ek olarak, daha önce bahsedilen arařtırma öncelikle erişilebilirliğe odaklanmış olsa da, yerleřtirme testi gibi görevlere görsel test yaklaşımları uygulanabilir. (Crispin, 2019)

4.1.5.1. Güvenlik Testi

Sızma testi olarak da bilinen güvenlik testlerine yardımcı olmak için makine öğrenimini kullanan kitaplar ve araçlar mevcuttur. Sızma testi, zayıf yönleri bulmak için bir sisteme girmeye çalışmayı içerir.

- **Kitaplar:** Sızma testi için makine öğrenimini nasıl kullanacağınızı öğreten kitaplar mevcuttur. Bunun bir örneğı, Chiheb Chebbi'nin yazdığı "Sızma Testi için Makine Öğreniminde Uzmanlaşmak: Python kullanarak kendi kendine öğrenme sistemlerini kırmak için kapsamlı bir beceri seti geliřtirin".
- **Araçlar:** Makine öğrenimini kullanan güvenlik testi araçları da vardır. Bazı örnekler arasında SEWorks'ün Pentoma'sı ve Wallarm'ın Wallarm Güvenlik Testi yer alır. (Melisha , 2018)

Örneğın Pentoma, birçok popüler bulut sunucusundaki web uygulamalarını tarayabildiğini ve aşağıdaki gibi güvenlik açıklarını bulabildiğini iddia ediyor:

- **SQL Enjeksiyonu:** Bu, bir veritabanından bilgi çalabilen bir saldırı türüdür.

- **Yerel Dosya Ekleme:** Bu, bir saldırganın sunucudaki dosyalara erişmesine olanak tanıyan bir saldırı türüdür.
- **Uzaktan Dosya Ekleme:** Bu, yerel dosya eklemeye benzer, ancak saldırganın başka bir sunucudaki dosyalara erişmesine olanak tanır.
- **Doğrulanmamış Yönlendirmeler ve İletmeler:** Bu, bir kullanıcıyı kötü amaçlı bir web sitesini ziyaret etmesi için kandırmak için kullanılabilir.
- **XSS (Siteler Arası Komut Dosyası Çalıştırma):** Bu, bir web sitesine kötü amaçlı kod yerleştirebilen bir saldırı türüdür.
- **Bilinmeyen Güvenlik Riskleri:** Pentoma ayrıca henüz bilinmeyen güvenlik risklerini de tespit edebildiğini iddia ediyor. (PENTOMA, 2018)

4.1.6. Genel Amaçlı Yapay Zeka Araçları ve Çerçevesi

Makine öğrenimi dünyasını derinlemesine inceleyen bu bölümde, Figure Eight'in araştırmasına dayanarak veri bilimcilerinin güvendiği en popüler araçlardan (çerçevesi adı verilen) bazıları inceleniyor. Bu araçlar, her biri veri bilimi iş akışında önemli bir rol oynayan özel araçlar gibi davranır.

- **Pandalar ve NumPy:** Bir kütüphane için veri düzenleyici olduğunuzu hayal edin. Popüler programlama dili Python için tasarlanan bu araçlar, steroidler üzerindeki dosya dolapları ve elektronik tablolar gibidir. Her şeyin iyi yapılandırılmış ve analize hazır olmasını sağlayarak verileri verimli bir şekilde yönetmemize ve işlememize yardımcı olurlar.
- **Scikit-learn:** Bu aracı ipuçlarını koklayan bir taziye olarak düşünün. Gizli kalıpları ve eğilimleri ortaya çıkarmak için verileri analiz etmede mükemmeldir. Scikit-learn, verilerimizden derlediği içgörülere dayanarak tahminlerde bile bulunabiliyor.

- **Matplotlib:** Hiç bir hava durumu haritasına bakıp hava tahminini anında anladınız mı? Bu araç, benzer çizelgeler ve grafikler oluşturmamıza olanak tanıyarak verilerimizi görselleştirmemizi ve karmaşık bilgilerden anlam çıkarmamızı kolaylaştırır.
- **TensorFlow ve Keras:** Bunlar makine öğrenimi dünyasının çekiçleri ve çivileridir. Makine öğrenimi modellerini oluşturmak ve eğitmek için kullanılan güçlü araçlardır. Bu modeller aslında maruz kaldıkları verilere dayanarak zaman içinde öğrenip gelişebilen bilgisayar programlarıdır.
- **Seaborn:** Bu araç, Matplotlib'in üzerine inşa edilerek ek özelleştirme seçenekleri sunar. Seaborn'u haritalarınız için dekorasyon seti olarak hayal edin. Yorumlanması ve başkalarıyla paylaşılması daha da kolay, görsel olarak çekici ve bilgilendirici grafikler oluşturmamıza yardımcı olur.
- **PyTorch ve Torch:** Bunlar, veri bilimcilerin çeşitli makine öğrenimi görevleri için sıklıkla kullandığı ek popüler çerçevelerdir.

İndirilebilir araçların ötesinde, Amazon Web Services (AWS), Google Cloud ve Microsoft Azure gibi büyük oyuncular tarafından sunulan çok çeşitli bulut tabanlı seçenekler bulunmaktadır. AWS Deep Learning AMI ve Google Cloud ML Engine gibi isimler taşıyan bu hizmetler, kendi bilgisayarlarımıza kurulum gerektirmeden güçlü makine öğrenimi araçlarına erişim sağlıyor. Bu, özellikle yerel makinemizin kaynaklarını zorlayabilecek büyük veri kümeleriyle çalışırken faydalı olabilir.

İlginç bir şekilde, Figure Eight tarafından yapılan araştırma, veri bilimcileri arasında, şirketler tarafından sunulanlara (ticari veya "beyaz etiketli" çözümler) kıyasla açık kaynaklı araçlara (ücretsiz olarak temin edilebilir) yönelik bir tercihi ortaya koyuyor. (Figure Eight, 2018) Bu, veri bilimi topluluğunun işbirlikçi ve açık doğasını vurgulamaktadır; burada bu açık kaynaklı araçların çoğu, daha önce tartıştığımız AGENT prototipini oluşturmak için kullanılan yapı taşlarıdır!

Temelde bu araçlar, veri bilimcilerine karmaşık sorunları çözme, gizli kalıpları ortaya çıkarma ve sonuçta verilerden değerli bilgiler elde etme konusunda güç verir. Makine

öğrenimi gelişmeye devam ettikçe, bu araç kutusunun daha da genişleyerek veri araştırması ve analizi için daha güçlü araçlar sunmasını bekleyebiliriz.

4.2. Vaka Çalışması: Python Kod Kalitesinin Otomatik Sınıflandırılması ve İyileştirilmesi için Makine Öğrenimi

4.2.1. Uygulama Fikri

Uygulama, Python kodunun kalitesini otomatik olarak sınıflandırmak ve iyileştirmek için makine öğrenimini kullanmayı amaçlamaktadır. Kod parçacıklarını hatalar açısından analiz ederek, geliştiricilerin daha az hatayla daha iyi kod yazmalarına yardımcı olan bir araç oluşturabiliriz. Bu, veri toplamayı, hataları tanınması için bir modeli eğitmeyi ve geliştiricilere geri bildirim vermeyi içerir.

4.2.2. Önem

Bu uygulama, otomatik kod kalitesi kontrollerine olan artan ihtiyacı karşıladığı için önemlidir. Manuel kod incelemesi yavaştır ve hatalar içerebilir. Bu işlemin otomatikleştirilmesi, kod incelemelerini daha hızlı ve daha doğru hale getirerek daha iyi yazılım ve daha verimli geliştirme sağlar.

4.2.3. Uygulama Etkisi

Uygulama, kod kalitesi hakkında anında geri bildirim sağlayarak geliştiricilerin sorunları erken düzeltmelerine ve hata ayıklamada zamandan tasarruf etmelerine yardımcı olur. Yalnızca yüksek kaliteli kodun dağıtılmasını sağlamak için CI/CD hattına entegre edilebilir.

4.2.4. Geleneksel Araçları Yapay Zeka ile Birleştirmenin Faydaları

Flake8 ve Pylint gibi araçları makine öğrenimiyle birlikte kullanmak çeşitli faydalar sunar:

1- Kapsamlı Analiz:

- **Geleneksel Araçlar:** Flake8 ve Pylint gibi araçlar kod stilini kontrol etmek ve yaygın hataları bulmak için harikadır.

- **Yapay Zeka Modelleri** : Makine öğrenimi, geleneksel araçların gözden kaçırabileceği daha derin, daha karmaşık sorunları bulabilir. Bu şekilde hem basit hem de karmaşık sorunlar bulunur ve düzeltilir.

2- Akıllı Geri Bildirim :

- **Geleneksel Araçlar** : Bu araçlar, belirlenen kurallara göre geri bildirim sağlar. Bu yararlıdır ancak tüm durumları kapsamayabilir.
- **Yapay Zeka Modelleri** : Yapay Zeka geçmiş hatalardan ve başarılarından ders çıkarır. Yapay Zeka, çok sayıda veriye bakarak daha doğru ve bağlam farkında geri bildirimler verir ve belirli kod için daha iyi öneriler sunar.

3- Verimlilik :

- **Geleneksel Araçlar** : Bu araçlar, ilk kontroller için hızlı ve güvenilirdir ve birçok yaygın sorunu hızla bulur.
- **Yapay Zeka Modelleri** : Yapay Zeka ile birleştirildiğinde, süreç daha da verimli hale gelir. Yapay zeka karmaşık analizleri ele alarak sistemin kod kalitesini hızlı ve güvenilir bir şekilde kontrol etmesini sağlar ve inceleme sürecini daha hızlı ve daha kapsamlı hale getirir.

4.2.5. Kod Verilerini Toplama, Analiz Etme ve Temizleme

Veri toplamadaki en büyük zorluklardan biri, kod hataları için büyük ve etiketli bir veri kümesinin olmamasıydı. Bunu aşmak için GitHub ve StackOverflow'dan indirilen yerel depolarından veri topladık ve çeşitli bir veri kümesi oluşturduk. Ayrıca farklı hata türleri ve doğru kod içeren kod örnekleri de ürettik. Hem toplanan hem de üretilen veriler, kodda daha sonra açıklanan araçlar kullanılarak sınıflandırıldı ve düzenlendi. Etiketlenmesi zaman alıcı olsa da bu ayrıntılı veri kümesi, etkili bir model eğitmek için çok önemliydi.

Kod Uygulamasını Başlatma

Bunu başarmak için kod uygulamamızdaki birkaç temel adımla başladık. Aşağıda, kullandığımız gerekli kitaplıklar ve araçlarla birlikte dahil olan ana adımların bir dökümü yer almaktadır :

Veri Toplama ve Analizi için Gerekli Kütüphaneleri İçer Aktarma

```
1 import os
2 import subprocess
3 import json
4 import tempfile
5 import concurrent.futures
6 import time
7 from stackapi import StackAPI
8 from bs4 import BeautifulSoup
9 import random
```

- **os:** Dosyaları okuma veya yazma gibi işletim sistemiyle etkileşime girmenize yardımcı olur.
- **subprocess:** Yeni süreçler başlatmanızı ve bunların girdilerini, çıktılarını ve hatalarını yönetmenizi sağlar.
- **json:** JSON (JavaScript Nesne Gösterimi) verileriyle çalışmak için kullanılır.
- **tempfile:** Geçici dosyalar ve dizinler oluşturur.
- **concurrent.futures:** Arka planda görevleri çalıştırmak için araçlar sağlar.
- **time:** Çeşitli zamanla ilgili işlevler sunar.
- **StackAPI:** Stack Exchange API ile çalışmak için bir kütüphane.
- **BeautifulSoup:** HTML ve XML belgelerini okumak ve analiz etmek için bir kütüphane.
- **random:** Farklı dağıtımlar için rastgele sayılar üretir.

Depo ve Dizinler Listesi

```
1 # List of repositories and directories
2 REPOS = [
3     ('TheAlgorithms/Python', 'Python-master'),
4     ('geekcomputers/Python', 'Python-master (1)'),
5     ('leachim6/hello-world', 'hello-world-main'),
6     ('labuladong/fucking-algorithm', 'algorithm-master'),
7     ('python/cpython', 'cpython-main'),
8 ]
```

REPOS: Her çiftin bir depo adı ve karşılık gelen bir dizin adı olduğu bir çift listesi. Bu, depoları yerel dizin adlarıyla eşleştirmek için kullanılır.

Eğitim ve test için seçilen GitHub depoları, hem doğru kod hem de hatalı kod dahil olmak üzere birçok farklı kod türüne sahipti. Bu, modelin çok çeşitli gerçek dünya kodlama durumlarını görmesini sağladı.

GitHub Depolarının Yerel Dizinlerinden Python Dosyalarını Toplama

```
1 def collect_local_code(repo_dir):
2     codes = []
3     for root, _, files in os.walk(repo_dir):
4         for file in files:
5             if file.endswith('.py'):
6                 codes.append(os.path.join(root, file))
7     return codes
```

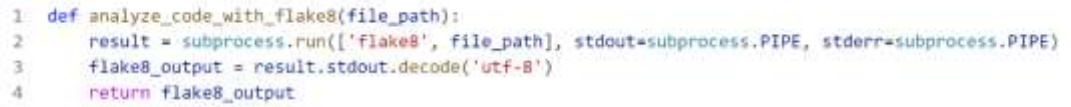
collect_local_code(repo_dir): Bu fonksiyon bir dizin yolunu argüman olarak alır. Dizin boyunca ilerler ve bulduğu tüm Python dosyalarına (.py) giden yolları toplar.

Flake8 ve Pylint Kullanarak Python Kodunu Analiz Etme

Flake8

Flake8, Python kodunun stilini ve kalitesini kontrol etmek için bir araçtır. Üç aracı birleştirir :

- **PyFlakes:** Kodda mantıksal hataları kontrol eder.
- **pycodestyle (formerly known as pep8):** Stil kılavuzunun uygulanmasını kontrol eder.
- **McCabe:** Kodun karmaşıklığını kontrol eder.



```
1 def analyze_code_with_flake8(file_path):
2     result = subprocess.run(['flake8', file_path], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
3     flake8_output = result.stdout.decode('utf-8')
4     return flake8_output
```

- **subprocess.run(['flake8', file_path], stdout=subprocess.PIPE, stderr=subprocess.PIPE):** Bu komut, belirtilen dosyada ('file_path') Flake8'i çalıştırır. Standart çıktıyı ve standart hata akışlarını yakalar.
- **result.stdout.decode('utf-8'):** Çıktıyı baytlardan UTF-8 dizesine çözer.
- **return flake8_output:** Flake8 analiz sonuçlarını içeren kodlanmış çıktıyı döndürür.

Pylint

Pylint, Python kodunun kalitesini kontrol etmek için kapsamlı bir araçtır. Şunları arar:

- Programlama hataları
- Bir kodlama standardının uygulanmasına yardımcı olur
- Kod kokularını koklar
- Basit yeniden düzenleme önerileri sunar

Pylint kapsamlı ve bazen aşırı katı olmasıyla bilinir, ancak projenin ihtiyaçlarına uyacak şekilde yapılandırılabilir.

```
1 def analyze_code_with_pylint(file_path):
2     result = subprocess.run(['pylint', file_path], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
3     pylint_output = result.stdout.decode('utf-8') + result.stderr.decode('utf-8')
4     return pylint_output
```

- **subprocess.run(['pylint', file_path], stdout=subprocess.PIPE, stderr=subprocess.PIPE):** Bu komut, Pylint'i belirtilen dosyada ('file_path') çalıştırır. Hem standart çıktıyı hem de standart hata akışlarını yakalar.
- **pylint_output = result.stdout.decode('utf-8') + result.stderr.decode('utf-8'):** Hem standart çıktıyı hem de standart hatayı baytlardan bir UTF-8 dizisine çözer ve bunları birleştirir.
- **return pylint_output:** Pylint analiz sonuçlarını içeren birleştirilmiş ve kodlanmış çıktıyı döndürür.

Farklılıkların Özeti

- **Kapsam :** Her iki araç da kod kalitesini kontrol ederken, Pylint daha kapsamlıdır, hataları kontrol eder, kodlama standartlarını uygular ve yeniden düzenlemeler önerir. Flake8 daha çok stile (pycodestyle aracılığıyla) ve temel mantıksal hatalara (PyFlakes aracılığıyla) odaklanır.
- **Çıktı :** Flake8 çıktısı genellikle daha kısadır ve stil sorunlarına odaklanırken, Pylint çıktısı daha ayrıntılıdır ve kod kalitesini iyileştirmeye yönelik öneriler içerir.

Analiz Çıktısına Göre Hataları Sınıflandırma

```
1 def classify_error(output):
2     if "syntax-error" in output:
3         return "Syntax Error"
4     elif "runtime-error" in output:
5         return "Runtime Error"
6     elif "logical-error" in output:
7         return "Logical Error"
8     elif "performance" in output:
9         return "Performance Issue"
10    elif "compatibility" in output:
11        return "Compatibility Issue"
12    elif "security" in output:
13        return "Security Issue"
14    else:
15        return "Correct Code"
```

- **classify_error(output):** Bu fonksiyon Pylint veya Flake8'den çıktıyı alır ve farklı hata türlerine göre sınıflandırır. Çıktıda belirli anahtar sözcükleri arar ve hata türünü tanımlayan bir dize döndürür.

Python Kodunu Toplamak ve Analiz Etmek İçin Depoları İşleme

```
1 def process_repository(repo):
2     repo_name, repo_dir = repo
3     code_files = collect_local_code(repo_dir)
4     repo_analysis = []
5
6     for file_path in code_files:
7         try:
8             with open(file_path, 'r', encoding='utf-8') as code_file:
9                 code = code_file.read()
10
11                 pylint_output = analyze_code_with_pylint(file_path)
12                 flake8_output = analyze_code_with_flake8(file_path)
13
14                 classification = classify_error(pylint_output + flake8_output)
15
16                 repo_analysis.append({
17                     'code': code,
18                     'classification': classification
19                 })
20
21         except Exception as e:
22             repo_analysis.append({
23                 'code': code,
24                 'classification': 'Error'
25             })
26
27     return repo_analysis
```

“process_repository” fonksiyonu aşağıdaki görevleri gerçekleştirir:

1. Belirtilen depo dizinindeki tüm Python dosyalarını toplar.
2. Her dosyayı okur ve “pylint” ve “flake8” kullanarak analiz eder.
3. Analiz sonuçlarını “pylint” ve “flake8”’in birleşik çıktısına göre sınıflandırır.
4. Dosya işleme sırasında herhangi bir istisnayı işler.
5. Her biri kodu ve sınıflandırmasını içeren bir analiz sonuçları listesi döndürür.

ThreadPoolExecutor ile Birden Fazla Depoyu Eşzamanlı Olarak İşleme

```
1 start_time = time.time()
2
3 all_codes = []
4 with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
5     future_to_repo = {executor.submit(process_repository, repo): repo for repo in REPOS}
6     for future in concurrent.futures.as_completed(future_to_repo):
7         try:
8             repo_analysis = future.result()
9             all_codes.extend(repo_analysis)
10        except Exception as e:
11            print(f"Error processing repository: {future_to_repo[future]}, error: {str(e)}")
```

- **‘all_codes = []’:** Tüm depolardan gelen analiz sonuçlarını depolamak için boş bir liste başlatır.
- **‘concurrent.futures.ThreadPoolExecutor(max_workers=4)’** Aynı anda 4 adede kadar iş parçacığını çalıştırabilen bir iş parçacığı havuzu yürütücüsü oluşturur. Bu, işlevin birden fazla depoyu aynı anda işlemesini sağlar.
- **‘executor.submit(process_repository, repo)’:** ‘process_repository’ işlevini, yürütme için yürütücüye bir argüman olarak depo tuple'ı (‘repo’) ile gönderir. İşlevin yürütülmesini temsil eden bir ‘Future’ nesnesi döndürür.
- **‘future_to_repo = {executor.submit(process_repository, repo): repo for repo in REPOS}’** repo for repo in REPOS}’ Anahtarların ‘Future’ nesneleri ve değerlerin karşılık gelen depo ikilileri olduğu bir sözlük oluşturur. Bu, her ‘Future’ ögesinin ilgili deposuna eşlenmesine yardımcı olur.
- **‘for future in concurrent.futures.as_completed(future_to_repo):’** Future nesneleri tamamlandıkça bunlar üzerinde yineleme yapar. ‘as_completed’ yöntemi, çalışmayı bitirdiklerinde Future nesnelerini üreten bir yineleyici döndürür.
- **‘future.result():** process_repository işlevinin çıktısı olan ‘Future’ nesnesinin sonucunu alır. İşlev bir istisna oluşturduysa, ‘result()’ ögesini çağırmak aynı istisnayı oluşturur.
- **all_codes.extend(repo_analysis):** Tamamlanan ‘Future’ ögesinden gelen analiz sonuçlarını ‘all_codes’ listesine ekler.

Summary

1. **Başlangıç Zamanını Kaydet :** Daha sonra toplam işlem süresini ölçmek için başlangıç zamanını yakalar.

2. **Boş Bir Liste Başlat** : Analiz sonuçlarını depolamak için boş bir liste oluşturur.
3. **Bir İş Parçacığı Havuzu Yürütücüsü Oluştur** : Eşzamanlı yürütme için en fazla 4 iş parçacığı içeren bir iş parçacığı havuzu kurar.
4. **Görevleri Gönder** : Her depo için 'process_repository' işlevini yürütücüye gönderir ve her 'Future'ı karşılık gelen deposuna eşler.
5. **Tamamlanan Future'ları İşle** : Tamamlanan 'Future' nesneleri üzerinde yineleme yapar, sonuçlarını alır ve bu sonuçlarla 'all_codes' listesini genişletir.
6. **İstisnaları İşle** : Bir deponun işlenmesi sırasında oluşan tüm istisnaları yakalar ve yazdırır.



Karmaşık Kod Örneklerini Oluşturma ve Sınıflandırma

```

1 # Generate random code samples
2 def generate_code_sample():
3     """Generate a random code sample for testing purposes.
4     Returns a string representing a code snippet.
5     """
6     # Randomly choose a function name
7     func_name = random.choice(['get_data', 'process_data', 'analyze_data'])
8     # Randomly choose a variable name
9     var_name = random.choice(['data', 'result', 'output'])
10    # Randomly choose a value
11    value = random.choice([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
12    # Randomly choose a data type
13    data_type = random.choice(['int', 'float', 'str', 'list', 'dict'])
14    # Randomly choose a file path
15    file_path = random.choice(['data.csv', 'output.txt', 'config.json'])
16    # Randomly choose a file name
17    file_name = random.choice(['data.csv', 'output.txt', 'config.json'])
18    # Randomly choose a file extension
19    file_ext = random.choice(['.csv', '.txt', '.json'])
20    # Randomly choose a file size
21    file_size = random.choice([1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432, 67108864, 134217728, 268435456, 536870912, 1073741824, 2147483648, 4294967296, 8589934592, 17179869184, 34359738368, 68719476736, 137438953472, 274877906944, 549755813888, 1099511627776, 2199023255552, 4398046511104, 8796093022208, 17592186044416, 35184372088832, 70368744177664, 140737488355328, 281474976710656, 562949953421312, 1125899906842624, 2251799813685248, 4503599627370496, 9007199254740992, 18014398509481984, 36028797018963968, 72057594037927936, 144115188075855872, 288230376151711744, 576460752303423488, 1152921504606846976, 2305843009213693952, 4611686018427387904, 9223372036854775808, 18446744073709551616, 36893488147419103232, 73786976294838206464, 147573952589676412928, 295147905179352825856, 590295810358705651712, 1180591620717411303424, 2361183241434822606848, 4722366482869645213696, 9444732965739290427392, 18889465931478580854784, 37778931862957161709568, 75557863725914323419136, 151115727451828646838272, 302231454903657293676544, 604462909807314587353088, 1208925819614629174706176, 2417851639229258349412352, 4835703278458516698824704, 9671406556917033397649408, 19342813113834066795298816, 38685626227668133590597632, 77371252455336267181195264, 154742504910672534362390528, 309485009821345068724781056, 618970019642690137449562112, 1237940039285380274899124224, 2475880078570760549798248448, 4951760157141521099596496896, 9903520314283042199192993792, 19807040628566084398385987584, 39614081257132168796771975168, 79228162514264337593543950336, 158456325028528675187087900672, 316912650057057350374175801344, 633825300114114700748351602688, 1267650600228229401496703205376, 2535301200456458802993406410752, 5070602400912917605986812821504, 10141204801825835211973625643008, 20282409603651670423947251286016, 40564819207303340847894502572032, 81129638414606681695789005144064, 162259276829213363391578010288128, 324518553658426726783156020576256, 649037107316853453566312041152512, 1298074214633706907132624082305024, 2596148429267413814265248164610048, 5192296858534827628530496329220096, 10384593717069655257060992658440192, 20769187434139310514121985316880384, 41538374868278621028243970633760768, 83076749736557242056487941267521536, 166153499473114484112975882535043072, 332306998946228968225951765070086144, 664613997892457936451903530140172288, 1329227995784915872903807060280344576, 2658455991569831745807614120560689152, 5316911983139663491615228241121378304, 10633823966279326983230456482242756608, 21267647932558653966460912964485513216, 42535295865117307932921825928971026432, 85070591730234615865843651857942052864, 170141183460469231731687303715884105728, 340282366920938463463374607431768211456, 680564733841876926926749214863536422912, 1361129467683753853853498429727072845824, 2722258935367507707706996859454145691648, 5444517870735015415413993718908291383296, 10889035741470030830827987437816582766592, 21778071482940061661655974875633165533184, 43556142965880123323311949751266331066368, 87112285931760246646623899502532662132736, 174224571863520493293247799005065324265472, 348449143727040986586495598010130648530944, 696898287454081973172991196020261297061888, 1393796574908163946345982392040522594123776, 2787593149816327892691964784081045188247552, 5575186299632655785383929568162090376495104, 11150372599265311570767859136324180752990208, 22300745198530623141535718272648361505980416, 44601490397061246283071436545296723011960832, 89202980794122492566142873090593446023921664, 178405961588244985132285746181186892047843328, 356811923176489970264571492362373784095686656, 713623846352979940529142984724747568191373312, 1427247692705959881058285969449495136382746624, 2854495385411919762116571938898990272765493248,
```

- **Hata Oluşturucular** : “generate_syntax_error”, “generate_runtime_error” vb. gibi işlevler, belirli türde hatalı kod parçacıkları oluşturur.
- **Doğru Kod Oluşturucu** : “generate_correct_code”, hatasız kod parçacıkları sağlar.
- **Örnek Oluşturucu** : “generate_complex_code_samples(num_samples)”, çeşitli hata türleri ve doğru kod dahil olmak üzere bir kod örneği karışımı oluşturur.
- **Entegrasyon** : Oluşturulan örnekler “all_codes”a eklenir ve işlev, oluşturmayı doğrulayan bir mesaj yazdırır.

Python Kodu için StackOverflow Verilerini Toplama, Analiz Etme ve Temizleme

```
1 # Collect StackOverflow data
2 SITE = StackAPI('stackoverflow')
3 SITE.max_pages = 25
4 SITE.page_size = 100
5
6 def extract_code(html_content):
7     soup = BeautifulSoup(html_content, 'html.parser')
8     code_snippets = [code.get_text() for code in soup.find_all('code')]
9     return "\n".join(code_snippets)
```

StackAPI: StackOverflow için StackAPI istemcisini başlatır.

- **'SITE.max_pages = 25':** Alınacak maksimum sayfa sayısını ayarlar.
- **'SITE.page_size = 100':** Sayfa başına öge sayısını ayarlar.

'extract_code(html_content)': HTML içeriğini ayrıştırır ve tüm kod parçacıklarını ayıklayarak bunları tek bir dize olarak döndürür.

- **'html_content':** Bu parametrenin HTML içeriği içeren bir dize olması beklenir. Bir StackOverflow sorusunun veya kod parçacıkları içerebilecek herhangi bir diğer HTML belgesinin gövdesini temsil eder.
- **'BeautifulSoup(html_content, 'html.parser')':** BeautifulSoup kitaplığı HTML içeriğini ayrıştırmak için kullanılır. HTML içeriğinden bir ayrıştırma ağacı oluşturur ve bu da kolay gezinme ve veri ayıklama olanağı sağlar. 'html.parser' argümanı kullanılacak ayrıştırıcıyı belirtir.
- **'soup.find_all('code')':** Bu yöntem HTML içeriğindeki tüm '<code>' etiketlerini bulur. Bu etiketler genellikle HTML belgelerindeki kod parçacıklarını kapatmak için kullanılır.
- **'[code.get_text() for code in soup.find_all('code')]'':** Bu liste kavrayışı, bulunan tüm '<code>' etiketleri üzerinde yineleme yapar ve her birinin metin içeriğini 'get_text()' yöntemini kullanarak çıkarır. Sonuç, her dizinin bir '<code>' etiketinin metin içeriği olduğu bir dizeler listesidir.

'return"\n".join(code_snippets)': Bu yöntem, çıkarılan tüm kod parçacıklarını tek bir dizeye birleştirir ve her parçacık bir yeni satır karakteriyle ('\n') ayrılır. Bu, toplanan kodu tek bir metin bloğu olarak işlemeyi kolaylaştırır.

```

1 def collect_stackoverflow_data(start_date, end_date):
2     questions = []
3     for page in range(1, SITE.max_pages + 1):
4         response = SITE.fetch('questions', tagged='python', #filter='withbody', sort='creation', order='desc', fromdate=start_date, todate=end_date, page=page)
5         questions.extend(response['items'])
6         if len(response['items']) < SITE.page_size:
7             break
8     time.sleep(2)

```

- **collect_stackoverflow_data(start_date, end_date)** : 'start_date' ve 'end_date' arasında oluşturulan 'python' etiketli StackOverflow sorularını toplar.
- **for page in range(1, SITE.max_pages + 1)** : Soruları almak için sayfalar arasında yineleme yapar.
- **response = SITE.fetch(...)** : Bir soru sayfası alır.
- **questions.extend(response['items'])** : Alınan soruları 'questions' listesine ekler.
- **if len(response['items']) < SITE.page_size** : Alınan öğe sayısı sayfa boyutundan azsa döngüyü kırar ve kullanılabilir verilerin sonunu belirtir.
- **time.sleep(2)** : Hız sınırlarına ulaşmamak için istekler arasında 2 saniye bekler.

StackOverflow Sorularından Kod Parçacıklarını İşleme

```
1 code_snippets = []
2 for question in questions:
3     if 'code' in question['body']:
4         code_content = extract_code(question['body'])
5         try:
6             with tempfile.NamedTemporaryFile(delete=False, suffix=".py") as temp_file:
7                 temp_file.write(code_content.encode('utf-8'))
8                 temp_file_path = temp_file.name
9
10                pylint_output = analyze_code_with_pylint(temp_file_path)
11                flake8_output = analyze_code_with_flake8(temp_file_path)
12
13                classification = classify_error(pylint_output + flake8_output)
14                os.remove(temp_file_path)
15            except Exception as e:
16                classification = 'Error'
17
18            code_snippets.append({
19                'code': code_content,
20                'classification': classification
21            })
22
23 return code_snippets
```

- **code_snippets = []**: Kod parçacıklarını ve sınıflandırmalarını depolamak için bir liste başlatır.
- **for question in questions**: Toplanan sorular üzerinde yineleme yapar.
- **if 'code' in question['body']**: Sorunun kod parçacıkları içerip içermediğini kontrol eder.
- **code_content = extract_code(question['body'])**: Sorunun gövdesinden kod parçacıklarını çıkarır.
- **with tempfile.NamedTemporaryFile(delete=False, suffix=".py") as temp_file**: Geçici bir Python dosyası oluşturur ve çıkarılan kodu ona yazar.
- **pylint_output = analyze_code_with_pylint(temp_file_path)**: Geçici dosyayı Pylint ile analiz eder.
- **flake8_output = analyze_code_with_flake8(temp_file_path)**: Geçici dosyayı Flake8 ile analiz eder.

- **classification = classify_error(pylint_output + flake8_output):** Kodu Pylint ve Flake8'in birleşik çıktısına göre sınıflandırır.
- **os.remove(temp_file_path):** Geçici dosyayı siler.
- **Exception Handling:** Analiz sırasında oluşan tüm istisnaları yakalar ve sınıflandırmayı 'Error' olarak ayarlar.
- **code_snippets.append(...):** Kod parçacığını ve sınıflandırmasını code_snippets listesine ekler.
- **return code_snippets:** Toplanan ve işlenen kod parçacıklarının sınıflandırmalarıyla birlikte listesini döndürür.

Belirli Tarih Aralıklarında StackOverflow Verilerini Toplamak İçin Kod

```

1  date_ranges = [
2      (1609459200, 1609545600),
3      (1609545600, 1609632000),
4  ]
5
6  for start_date, end_date in date_ranges:
7      stackoverflow_data = collect_stackoverflow_data(start_date, end_date)
8      all_codes.extend(stackoverflow_data)
9      time.sleep(10)

```

- **date_ranges :** StackOverflow verilerini toplamak için tarih aralıklarını belirtir (Unix zaman damgalarında).
- **for start_date, end_date in date_ranges:** Her tarih aralığını yineler.
- **stackoverflow_data = collect_stackoverflow_data(start_date, end_date):** Belirtilen tarih aralığı için veri toplar..
- **all_codes.extend(stackoverflow_data):** Toplanan verileri 'all_codes' listesine ekler.
- **time.sleep(10):** Oran sınırlarına ulaşmamak için istekler arasında 10 saniye bekler.

Toplanan Verileri Kaydetme

```
1 # Clean the collected data
2 cleaned_code_data = [entry for entry in all_codes if entry['code'].strip()]
3
4 with open("cleaned_training_data_Final.json", "w", encoding="utf-8") as f:
5     json.dump(cleaned_code_data, f, indent=4)
6
7 end_time = time.time()
8 execution_time = end_time - start_time
9 print(f"Data collection and cleaning complete. Cleaned data saved to cleaned_training_data_Final.json")
10 print(f"Total execution time: {execution_time} seconds")
```

1. **Verileri Temizle** : “all_codes”tan boş veya yalnızca boşluk içeren kod parçacıkları içeren girdileri filtreler.
2. **Temizlenen Verileri Kaydet** : Temizlenen verileri uygun biçimlendirmeyeyle bir JSON dosyasına yazar.
3. **Yürütme Süresini Hesapla** : Tüm süreç için geçen toplam süreyi ölçer.
4. **Mesajları Yazdır** : Tamamlama ve yürütme süresi mesajlarını konsola çıktı olarak verir.

4.2.6. Veri Hazırlama ve Model Eğitim İş Akışı

- **Kullanılan Model ve Özellikleri**

Bu uygulamada, metin sınıflandırması için XGBoost modelini kullandık. XGBoost, sınıflandırma ve tahmin görevleri için güçlü ve popüler bir makine öğrenimi modelidir. Sıralı karar ağaçları oluşturarak performansı iyileştirmek için Gradient Boosting tekniklerine güvenir; burada her ağaç bir öncekinin hatalarını düzeltir.

- **XGBoost'un Avantajları**

- **Yüksek Verimlilik** : XGBoost, büyük miktarda veriyi hızlı ve verimli bir şekilde işleyebilir ve bu da onu hızlı yanıt süreleri gerektiren pratik uygulamalar için ideal hale getirir.
- **Yüksek Doğruluk** : XGBoost, gradyan artırma tekniklerini kullanarak doğruluğunu sürekli olarak iyileştirebilir ve bu da yüksek kaliteli sınıflandırma sonuçlarına yol açabilir.

- **Esneklik :** XGBoost, birçok veri türünü ve biçimini destekler ve bu da onu farklı alanlardaki çeşitli uygulamalar için uygun hale getirir.
- **Etkileşim İşleme:** Model, verilerdeki farklı özellikler arasındaki karmaşık etkileşimleri etkili bir şekilde işleyebilir.
- **Paralel İşleme :** XGBoost, çoklu işlemcili sistemlerde eğitim sürecini hızlandıran paralel işlemeyi destekler.

- **XGBoost'u Seçme**

XGBoost'u seçmeden önce, Random Forest ve Neural Networks gibi birkaç başka modeli test ettik. Ancak, XGBoost doğruluk ve verimlilik açısından en iyi sonuçları gösterdi. Ek olarak, XGBoost eğitim sürecinde diğer modellere kıyasla daha hızlıydı ve bu da gerektiğinde yeniden eğitimi hızlandırmaya yardımcı oluyor.

- **XGBoost Nasıl Çalışır**

XGBoost (Extreme Gradient Boosting), daha zayıf modellerden oluşan bir grubu birleştirerek güçlü bir model oluşturmak için Gradient Boosting tekniklerine dayanan bir modeldir. Bu model, temel modeller olarak çoğunlukla karar ağaçlarını kullanır. İşte XGBoost'un nasıl çalıştığına dair ayrıntılı bir açıklama:

1-Sıralı Karar Ağaçları :

- XGBoost, karar ağaçlarını sıralı olarak oluşturur. Bir karar ağacıyla (zayıf bir model) başlar ve ardından her bir sonraki ağaç, önceki ağaç tarafından yapılan hatalara göre iyileştirilir. Bu işlem, belirli sayıda ağaç oluşturulana kadar birkaç kez tekrarlanır.

2-Kayıp Hesaplama ve Modeli İyileştirme :

- Her adımda, model mevcut model tarafından yapılan hataları ölçen bir kayıp fonksiyonu hesaplar. XGBoost, kayıp fonksiyonunun değerini azaltmak için Gradient Descent gibi sayısal optimizasyon tekniklerini kullanır.

3- Örneklerin Ağırlıklandırılması :

- Her yinelemeden sonra, veri kümesindeki örneklerin ağırlıkları model tarafından yapılan hatalara göre ayarlanır. Yanlış sınıflandırılan örneklere daha yüksek ağırlıklar verilirken, doğru sınıflandırılan örneklere daha düşük ağırlıklar verilir. Bu, modelin sonraki yinelemelerde hataları düzeltmeye odaklanmasına yardımcı olur.

4- Ağaçları Budama :

- XGBoost, performansı önemli ölçüde iyileştirmeyen dalları kaldırarak ağaçları budamak için teknikler kullanır. Bu, karmaşıklığı azaltmaya ve aşırı uyumu önlemeye yardımcı olur.

5- Düzenleme :

- Genelleştirmeyi iyileştirmek ve aşırı uyumu önlemek için XGBoost, L1 (Lasso) ve L2 (Ridge) gibi düzenleme tekniklerini kullanır. Bu teknikler, aşırı karmaşıklığı önlemek için modelin parametrelerine kısıtlamalar ekler.

6- Paralel ve Dağıtılmış İşleme :

- XGBoost, paralel işlemeyi destekler ve birçok hesaplamanın aynı anda gerçekleştirilmesine olanak tanıyarak eğitim sürecini hızlandırır. Eğitim, süreci daha da hızlandırmak için birden fazla makineye dağıtılabilir.

• Özellik Çıkarımı

Herhangi bir makine öğrenimi modeli eğitiminde, özellik çıkarma, ham verileri modelin anlayabileceği ve öğrenme için kullanabileceği bir forma dönüştürmek için esastır. Önceki kodda, metin verilerini sayısal özelliklere dönüştürmek için TF-IDF (Terim Frekansı-Ters Belge Frekansı) kullandık.

Kodda Özellik Çıkarımı Adımları :

1- Verileri Hazırlama :

- Yüklenen verilerden önemli alanları, yani kodları (metinleri) ve sınıflandırmaları çıkardık.

2- TF-IDF Kullanarak Metin Verilerini Sayısal Özelliklere Dönüştürme :

- TfidfVectorizer: Bu sınıftan bir nesne oluşturduk ve bunu kod listesine uyguladık. Metinleri analiz eder, bireysel kelimeleri (token'ları) belirler, her bir kelimenin her belgedeki sıklığını ve TF-IDF kullanarak tüm belgelerdeki önemini hesaplar. Son olarak, metni sayısal bir özellik matrisine dönüştürür.

Kod Uygulamasını Başlatma

Gerekli Kütüphaneleri İçe Aktarma

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
4 import xgboost as xgb
5 import joblib
6 import matplotlib.pyplot as plt
7 import seaborn as sns
```

- **TfidfVectorizer:** Ham belgelerin bir koleksiyonunu TF-IDF özelliklerinin bir matrisine dönüştürür.
- **train_test_split:** Dizileri veya matrisleri rastgele eğitim ve test alt kümelerine böler.
- **classification_report, accuracy_score, confusion_matrix:** Sınıflandırma performansını değerlendirmek için ölçümler.
- **xgboost:** Gradyan artırma algoritmaları için bir kütüphane.
- **joblib:** Python nesnelerini (modellerini) kaydetmek ve yüklemek için bir kütüphane.
- **matplotlib.pyplot, seaborn:** Veri görselleştirme için kütüphaneler.

Veri Yükleme

```
1 def load_data():
2     print("Loading data...")
3     with open('cleaned_training_data_Final.json', 'r', encoding='utf-8') as f:
4         data = json.load(f)
5     print("Data loaded.")
6     return data
```

`load\_data` işlevi, bir JSON dosyasından bir veri kümesini yüklemekten sorumludur. Veri yükleme işleminin başladığını belirten bir mesaj yazdırarak başlar. Ardından, 'cleaned_training_data_Final.json' adlı dosyayı UTF-8 kodlamasıyla okuma modunda açar. `json.load` metodunu kullanarak, dosyanın içeriğini `data` adlı bir değişkene okur. Verileri başarıyla yükledikten sonra, bir onay mesajı yazdırır. Son olarak, fonksiyon yüklenen verileri daha fazla işleme veya analiz için döndürür.

Verilerin Ön İşlenmesi

```
1 def preprocess_data(data):
2     codes = [entry['code'] for entry in data]
3     classifications = [entry['classification'] for entry in data]
4     return codes, classifications
```

`preprocess\_data` fonksiyonu, bir veri kümesinden belirli alanları çıkarmak ve bunları ayrı listeler halinde düzenlemek için tasarlanmıştır. Bir sözlük listesi olması gereken `data` adlı bir argüman alır. Her sözlüğün 'code' ve 'classification' anahtarları vardır. Fonksiyon, veri kümesindeki her sözlüğü inceleyerek 'code' ve 'classification' anahtarları için değerleri alır. Bu değerler `codes` ve `classifications` adlı iki ayrı listede saklanır. Son olarak, fonksiyon bu iki listeyi döndürür.

Verileri Vektörize Etme

```
1 def vectorize_data(codes):
2     vectorizer = TfidfVectorizer()
3     X = vectorizer.fit_transform(codes)
4     return X, vectorizer
```

`vectorize\_data` fonksiyonu, TF-IDF (Terim Frekansı-Ters Belge Frekansı) yöntemini kullanarak bir metin verisi (kodları) listesini sayısal vektörlere dönüştürür. Bir `TfidfVectorizer` nesnesi oluşturarak başlar ve ardından verilen kodlara uydurarak bunları bir TF-IDF özellikleri matrisine dönüştürür. Fonksiyon, daha sonra kullanılmak üzere sonuç matrisi `X` ve `vectorizer` nesnesini döndürür. Bu adım, metni modellerin

anlayabileceği ve işleyebileceği sayısal bir biçime dönüştürerek makine öğrenimi modelleri için metin verilerini hazırlamak için önemlidir.

Verileri Bölme

```
1 def split_data(X, y):  
2     return train_test_split(X, y, test_size=0.2, random_state=42)
```

`split\_data` işlevi, veri kümesini eğitim ve test kümelerine ayırır. İki bağımsız değişken alır: `X` (özellikler) ve `y` (etiketler). Scikit-learn'den `train\_test\_split` yöntemini kullanarak, verileri %20'si test için ve %80'i eğitim için olacak şekilde böler ve sonuçların yeniden üretilebilmesini sağlamak için sabit 42 rastgele durumu kullanır. Bu işlev, doğrulama için ayrı bir veri kümesi sağlayarak makine öğrenimi modellerinin performansını değerlendirmek için önemlidir.

Etiket Kodlaması

```
1 def label_encoding(y):  
2     label_to_index = {label: index for index, label in enumerate(set(y))}  
3     index_to_label = {index: label for label, index in label_to_index.items()}  
4     y_encoded = [label_to_index[label] for label in y]  
5     return y_encoded, label_to_index, index_to_label
```

`label\_encoding` işlevi, kategorik etiketleri sayısal değerlere dönüştürür. Etiketlerin bir listesi olan `y` adlı bir bağımsız değişken alır. Fonksiyon ilk olarak `enumerate` fonksiyonunu kullanarak her benzersiz etiketten bir dizine eşleme oluşturur ve bunu `label\_to\_index` sözlüğünde depolar. Daha sonra `index\_to\_label` sözlüğünde depolanan dizinden etikete ters bir eşleme oluşturur. `y` içindeki etiketler daha sonra `label\_to\_index` kullanılarak karşılık gelen sayısal değerlere kodlanır ve `y\_encoded` ile sonuçlanır. Son olarak, fonksiyon `label\_to\_index` ve `index\_to\_label` eşlemeleriyle birlikte kodlanmış etiketler `y\_encoded` döndürür. Bu kodlama, kategorik etiketleri makine öğrenimi algoritmalarının işleyebileceği bir biçime dönüştürmek için önemlidir.

Eğitim Modeli

```
1 def train_model(X_train, y_train):
2     model = xgb.XGBClassifier(eval_metric='mlogloss')
3     model.fit(X_train, y_train)
4     return model
```

`train\_model` fonksiyonu, sağlanan eğitim verileri üzerinde bir XGBoost sınıflandırıcısı eğitir. İki argüman alır: `X\_train` (eğitim veri kümesinin özellikleri) ve `y\_train` (eğitim veri kümesinin etiketleri). Fonksiyon, değerlendirme metriği 'mlogloss' (çok sınıflı logaritmik kayıp) olarak ayarlanmış bir XGBoost sınıflandırıcısını başlatarak başlar; bu, sınıflandırıcının çok sınıflı sınıflandırma problemlerindeki performansını değerlendirmeye yardımcı olur. Daha sonra modeli eğitim verilerine uydurur ve modelin sağlanan örneklerden öğrenmesini sağlar. Son olarak, fonksiyon eğitilmiş modeli döndürür, böylece tahminlerde bulunmak veya daha fazla değerlendirme yapmak için kullanılabilir. Bu fonksiyon, belirli bir veri kümesi üzerinde bir XGBoost sınıflandırıcısını eğitmenin basit bir yoludur.

Modeli Kaydetme

```
1 def save_model(model, vectorizer, label_to_index):
2     joblib.dump(model, 'xgb_model.pkl')
3     joblib.dump(vectorizer, 'vectorizer.pkl')
4     joblib.dump(label_to_index, 'label_to_index.pkl')
5     print("Model, vectorizer, and label_to_index saved.")
```

`save\_model` fonksiyonu, Joblib kütüphanesini kullanarak eğitilmiş modeli, vektörleştiriciyi ve etiket-indeks eşlemesini diske kaydetmek için tasarlanmıştır. Üç argüman alır: `model` (eğitilmiş XGBoost modeli), `vectorizer` (TF-IDF vektörleştiricisi) ve `label\_to\_index` (etiket-indeks eşlemesi). İşlev bu nesneleri serileştirir ve bunları ayrı dosyalara kaydeder: model için 'xgb_model.pkl', vektörleştirici için 'vectorizer.pkl' ve

etiket-indeks eşlemesi için 'label_to_index.pkl'. Son olarak, nesnelerin başarıyla kaydedildiğini doğrulayan bir mesaj yazdırır. Bu işlev, eğitilen bileşenlerin korunması, bunların yeniden yüklenmesi ve modeli yeniden eğitmeye gerek kalmadan tahminler veya daha ileri analizler için kullanılması için önemlidir.

Eğitim Bilgilerini Yazdırma

```
1 def print_training_info(X_train, X_test, y_train, y_test, model, index_to_label):
2     y_test_encoded, _ = label_encoding(y_test)
3     y_pred = model.predict(X_test)
4     accuracy = accuracy_score(y_test_encoded, y_pred)
5     print(f"Model accuracy: {accuracy}")
6     print("Classification report:")
7     print(classification_report(y_test_encoded, y_pred))
8
9
10    cm = confusion_matrix(y_test_encoded, y_pred)
11    plt.figure(figsize=(10, 7))
12    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=index_to_label.values(), yticklabels=index_to_label.values())
13    plt.xlabel('Predicted')
14    plt.ylabel('Actual')
15    plt.title('Confusion Matrix')
16    plt.show()
```

`print\_training\_info` işlevi, eğitilen bir modelin performansını değerlendirir ve görselleştirir. Birkaç argüman alır: `X\_train`, `X\_test`, `y\_train`, `y\_test`, `model` ve `index\_to\_label`. İşlev, test etiketlerini `label\_encoding` işlevini kullanarak kodlayarak başlar. Daha sonra, test kümesi için etiketleri tahmin etmek için modeli kullanır ve modelin doğruluğunu hesaplar. Doğruluk puanı ve ayrıntılı bir sınıflandırma raporu, modelin performansını göstermek için yazdırılır.

Daha sonra, işlev, modelin her sınıfı tahmin etmedeki performansını görselleştirmek için bir karışıklık matrisi oluşturur. Karışıklık matrisini uygun etiketler ve bir başlıkla çizmek için Seaborn'un ısı haritasını kullanır. Çizim, modelin performansının görsel bir temsilini vermek için görüntülenir. Bu işlev, modelin eğitim verilerinden ne kadar iyi öğrendiğini ve yeni verileri ne kadar doğru tahmin edebildiğini anlamak için önemlidir.

Ana İşlev

```
1 def main():
2     data = load_data()
3     codes, classifications = preprocess_data(data)
4     X, vectorizer = vectorize_data(codes)
5     X_train, X_test, y_train, y_test = split_data(X, classifications)
6     y_train_encoded, label_to_index, index_to_label = label_encoding(y_train)
7     model = train_model(X_train, y_train_encoded)
8     save_model(model, vectorizer, label_to_index)
9     print_training_info(X_train, X_test, y_train, y_test, model, index_to_label)
10
11 if __name__ == "__main__":
12     main()
```

`main` işlevi, verileri yüklemekten ve ön işleme tabi tutmaktan modeli eğitmeye ve değerlendirmeye kadar tüm makine öğrenimi iş akışını düzenler. `load\_data` işlevini kullanarak verileri yükleyerek başlar. Daha sonra veriler, gerekli alanları çıkarmak için `preprocess\_data` ile ön işlenir. `vectorize\_data` işlevi, metin verilerini TF-IDF vektörleştirmesini kullanarak sayısal vektörlere dönüştürür.

Daha sonra, veriler `split\_data` işlevi kullanılarak eğitim ve test kümelerine ayrılır. Eğitim etiketleri `label\_encoding` ile sayısal değerlere kodlanır. XGBoost modeli, `train\_model` işlevi kullanılarak eğitim verileri üzerinde eğitilir. Eğitildikten sonra, model, vektörleştirici ve etiket-indeks eşlemesi `save\_model` işlevi ile diske kaydedilir.

Son olarak, modelin performansını değerlendirmek ve sonuçları görselleştirmek için `print\_training\_info` işlevi çağrılır. Tüm süreç, betik doğrudan çalıştırıldığında `main` fonksiyonunu çağırarak başlar.

Bu tam fonksiyon, makine öğrenimi hattının her adımının doğru sırayla yapılmasını sağlayarak ham verilerden model değerlendirmesine ve kaydedilmesine kadar sorunsuz bir iş akışı sağlar.

4.2.7. Test İçin Kod Yükleme ve Hazırlama

Model Yükleme

```
1 def load_model():
2     model = joblib.load('xgb_model.pkl')
3     vectorizer = joblib.load('vectorizer.pkl')
4     label_to_index = joblib.load('label_to_index.pkl')
5     index_to_label = {index: label for label, index in label_to_index.items()}
6     return model, vectorizer, index_to_label
```

'load_model' fonksiyonu, bir makine öğrenimi modelinin daha önce kaydedilmiş parçalarını diskten yüklemekten sorumludur. Joblib kütüphanesini kullanarak eğitilmiş modeli, TF-IDF vektörleştiriciyi ve etiket-indeks eşlemesini alır. Özellikle, modeli 'xgb_model.pkl'den, vektörleştiriciyi 'vectorizer.pkl'den ve etiket-indeks eşlemesini 'label_to_index.pkl'den yükler. Ayrıca, sayısal etiketleri orijinal kategorilerine geri çözmeye yardımcı olmak için yüklenen 'label_to_index'ten ters bir eşleme, 'index_to_label' oluşturur. Son olarak, fonksiyon yüklenen modeli, vektörleştiriciyi ve indeks-etiket eşlemesini daha fazla kullanım için döndürür.

Test Kodunu Okuma

```
1 def read_test_code(file_path):
2     with open(file_path, 'r', encoding='utf-8') as f:
3         return f.read()
```

'read_test_code' fonksiyonu bir metin dosyasının içeriğini okur. Okunacak dosyanın yolunu belirten 'file_path' adlı bir argüman alır. Fonksiyon dosyayı UTF-8 kodlamasıyla okuma modunda açar ve 'f.read()' metodunu kullanarak tüm içeriğini okur. Dosyanın içeriği daha sonra bir dize olarak döndürülür.

Kodu Tek Satırlık Bloklara Dönüştürme

```
1 def convert_code_to_single_line_blocks(code):
2     lines = code.split('\n')
3     blocks = []
4     current_block = []
5     inside_block = False
6
7     for line in lines:
8         stripped_line = line.rstrip()
9         if stripped_line.startswith('#'):
10             if current_block:
11                 blocks.append("\n".join(current_block))
12                 current_block = []
13                 blocks.append(stripped_line)
14                 inside_block = False
15             else:
16                 current_block.append(stripped_line)
17                 inside_block = True
18
19     if current_block:
20         blocks.append("\n".join(current_block))
21
22     return blocks
```

`convert\_code\_to\_single\_line\_blocks` fonksiyonu bir kod bloğunu işler ve onu tek satırlık bloklara düzenler. Birden fazla kod satırı içeren bir dize olan `code` adlı bir argüman alır. Fonksiyon giriş kodunu ayrı satırlara böler ve her satırı inceler.

- Bir satır bir yorum karakteriyle (`#`) başlıyorsa, geçerli kod bloğunu (varsa) `blocks` listesine ekler ve `current\_block`'u sıfırlar. Daha sonra yorum satırını ayrı bir blok olarak ekler.
- Satır bir yorumla başlamıyorsa, satırı `current\_block`'a ekler.

Yinelemenin sonunda, `current\_block`'ta kalan satırlar varsa, bu bloğu `blocks` listesine ekler. Son olarak, fonksiyon her bloğun tek satırlık bir kod dizesi olduğu `blocks` listesini döndürür.

Bu fonksiyon, özellikle kodla programatik olarak çalışırken veya kodu analiz ederken, bir kod bloğunu daha küçük, daha yönetilebilir parçalara ayırmak için kullanışlıdır.

Son Satırları Kaldırma

```
1 def remove_trailing_newlines(blocks):
2     cleaned_blocks = []
3     for block in blocks:
4         cleaned_block = block.rstrip('\n')
5         cleaned_blocks.append(cleaned_block)
6     return cleaned_blocks
```

`remove\_trailing\_newlines` fonksiyonu, kod blokları listesinden son satır karakterlerini kaldırır. Her dizenin bir kod bloğu olduğu bir dize listesi olan `blocks` adlı bir argüman alır. Fonksiyon, giriş listesindeki her bloğu inceler ve son satır karakterlerini kaldırmak için `rstrip` metodunu kullanır. Temizlenen blok daha sonra `cleaned\_blocks` adlı yeni bir listeye eklenir. Fonksiyon, tüm blokları işledikten sonra `cleaned\_blocks` listesini döndürür. Bu fonksiyon, her bloğun sonunda fazladan yeni satır karakteri olmadığından emin olarak kod bloklarını temizlemek için kullanışlıdır ve kodun daha kolay çalışılmasını ve okunmasını sağlar.

Kod Bloklarını Sınıflandırma

```
1 def classify_blocks(model, vectorizer, index_to_label, blocks):
2     classifications = []
3     for block in blocks:
4         features = vectorizer.transform([block])
5         prediction_index = model.predict(features)
6         prediction_label = index_to_label[prediction_index[0]]
7         classifications.append(prediction_label)
8     return classifications
```

`classify\_blocks` fonksiyonu, önceden eğitilmiş bir makine öğrenimi modeli ve bir vektörleştirici kullanarak bir kod bloğu listesini sınıflandırır. Dört argüman alır: `model` (eğitilmiş model), `vectorizer` (TF-IDF vektörleştirici), `index\_to\_label` (sayısal endekslerden karşılık gelen etiketlere eşleme) ve `blocks` (sınıflandırılacak kod blokları listesi).

İşlev aşağıdaki adımları gerçekleştirir :

1. Boş bir liste `classifications` başlatın: Bu, sınıflandırma sonuçlarını depolayacaktır.
2. `blocks` listesindeki her blok üzerinde yineleme yapın:
 - **Bloğu dönüştürün** : Bloğu bir özellik vektörüne dönüştürmek için `vectorizer`'ı kullanın.
 - **Bir tahmin alın** : Sayısal bir endeks olan bir tahmin elde etmek için özellik vektörünü `model`'e geçirin.
 - **Etikete dönüştürün** : Sayısal endeksi karşılık gelen etikete değiştirmek için `index\_to\_label` eşlemesini kullanın.
 - **Sonucu depolayın** : Tahmin edilen etiketi `classifications` listesine ekleyin.

Son olarak, işlev sınıflandırma sonuçlarının listesini döndürür. Bu, eğitilmiş bir modeli yeni verilere uygulamaya yardımcı olur ve her kod bloğunun eğitim sırasında öğrenilen kalıplara göre sınıflandırılmasını sağlar.

Sınıflandırma Sonuçlarını Yazdırma

```
1 def print_classification_results(classifications):
2     result_counts = {label: {'count': 0, 'indices': []} for label in set(classifications)}
3     for i, label in enumerate(classifications, 1):
4         result_counts[label]['count'] += 1
5         result_counts[label]['indices'].append(i)
6
7     for label, info in result_counts.items():
8         print(f"{label}: {info['count']} times in functions {info['indices']}")
```

`print\_classification\_results` işlevi, her etiketin oluşumlarını sayarak ve her etiketin olduğu dizinleri listeleterek sınıflandırma sonuçlarının bir özetini sağlar. Tahmin edilen etiketlerin bir listesi olan `classifications` adlı bir argüman alır.

İşlev, her etiket için sayıyı ve dizinleri depolamak üzere `result\_counts` adlı bir sözlüğü başlatır. `classifications` listesini inceleyerek etiketleri ve dizinlerini numaralandırır. Her etiket için sayıyı artırır ve dizini `result\_counts` içindeki karşılık gelen girdiye ekler.

İşlev, tüm sınıflandırmaları işledikten sonra, her etiket için oluşum sayısını ve etiketin tahmin edildiği dizinleri gösteren bir özet yazdırır. Bu işlev, tahmin edilen etiketlerin dağılımını analiz etmeye ve her etiketin sınıflandırma sonuçlarında ne sıklıkta görüldüğünü anlamaya yardımcı olur.

4.2.8. Veri Yönetimi Eğitimi ve Yeniden Eğitim İş Akışı

- **Veri Yönetimi**
 - **Verileri Birleştir ve Karıştır** : Veri kümelerini birleştirir, yinelenenleri kaldırır ve verileri karıştırır.
 - **Verileri Kaydet** : Veri kümesini bir JSON dosyasına kaydeder.
 - **Sınıflandırma Girişini Ayrıştır** : Sınıflandırma verilerini içeren bir JSON dosyasını okur ve ayrıştırır.
 - **Sınıflandırmaları Güncelle** : Kod bloklarının listesini doğru sınıflandırmalarıyla günceller.
- **Eğitim ve Yeniden Eğitim**
 - **Eğitim Bilgilerini Yazdır** : Model performansını değerlendirir ve görselleştirir.
 - **Modeli Yeniden Eğit** : Modeli güncellenmiş verilerle yeniden eğitir ve güncellenmiş modeli kaydeder.
- **Ana İş Akışı**
 - **Test Et ve Yeniden Eğit** : Modeli yeni kod üzerinde test eder, isteğe bağlı olarak sınıflandırmaları günceller ve kullanıcı girdisine göre modeli yeniden eğitir.

Kod Uygulamasını Başlatma

Verileri Birleştirme ve Karıştırma

```
1 def merge_and_shuffle_data(original_data, new_data):
2     codes_in_original = {entry['code'] for entry in original_data}
3     merged_data = original_data[:]
4     for entry in new_data:
5         if entry['code'] not in codes_in_original:
6             merged_data.append(entry)
7     random.shuffle(merged_data)
8     return merged_data
```

`merge\_and\_shuffle\_data` fonksiyonu iki veri kümesini birleştirir, yinelenenleri kaldırır ve ortaya çıkan veri kümesini karıştırır. İki argüman alır: `original\_data` (ilk veri kümesi) ve `new\_data` (birleştirilecek veri kümesi).

İlk olarak, fonksiyon `original\_data`'dan 'kod' değerlerini çıkarır ve bunları `codes\_in\_original` kümesinde depolar. Bu küme, yinelenenleri hızlı bir şekilde kontrol etmeye yardımcı olur. `merged\_data` listesi `original\_data`'nın bir kopyasıyla başlatılır.

Fonksiyon daha sonra `new\_data`'daki her girişi inceler. Girişin 'kod' değeri `codes\_in\_original`'da zaten yoksa, giriş `merged\_data`'ya eklenir.

Birleştirmeden sonra, fonksiyon rastgele bir sıra sağlamak için `merged\_data` listesini karıştırır. Son olarak, `merged\_data` listesini döndürür. Bu işlem, yinelenenleri kaldırırken veri kümelerini birleştirmek ve verilerin rastgele sıralandığından emin olmak için kullanışlıdır.

Verileri Kaydetme

```
1 def save_data(data, file_path='cleaned_training_data_Final.json'):
2     with open(file_path, 'w', encoding='utf-8') as f:
3         json.dump(data, f, ensure_ascii=False, indent=4)
4     print("Updated data saved.")
```

`save\_data` işlevi bir veri kümesini bir JSON dosyasına kaydeder. İki argüman alır: kaydedilecek veri kümesi olan `data` ve dosyanın konumunu belirten ve varsayılan olarak `cleaned\_training\_data\_Final.json` olan isteğe bağlı bir `file\_path`.

İşlev belirtilen dosyayı UTF-8 kodlamasıyla yazma modunda açar. `json.dump` yöntemini kullanarak veri kümesini dosyaya yazar. `ensure\_ascii=False` parametresi ASCII olmayan karakterlerin düzgün kodlanmasını sağlar ve `indent=4` JSON çıktısının okunabilirlik için 4 boşluk girintisiyle biçimlendirilmesini sağlar. Son olarak, işlem güncellenen verilerin kaydedildiğini belirten bir onay mesajı yazdırır.

Sınıflandırma Girişini Ayrıştırma

```
1 def parse_classification_input(file_path):
2     with open(file_path, 'r', encoding='utf-8') as f:
3         correct_classifications = json.load(f)
4
5     if not isinstance(correct_classifications, dict):
6         raise ValueError("Correct classifications should be a dictionary.")
7
8     return correct_classifications
```

`parse\_classification\_input` işlevi, sınıflandırma verilerini içeren bir JSON dosyasını okur ve ayrıştırır ve ayrıştırılan verilerin bir sözlük olduğundan emin olur. JSON dosyasına giden yolu belirten bir argüman olan `file\_path` alır.

İşlev, dosyayı UTF-8 kodlamasıyla okuma modunda açar ve içeriğini `correct\_classifications`'a ayrıştırmak için `json.load` yöntemini kullanır. Veriler yüklendikten sonra, işlev `correct\_classifications`'ın bir sözlük olup olmadığını kontrol eder. Değilse, doğru sınıflandırmaların bir sözlük olması gerektiğini belirten bir `ValueError` oluşturur. Son olarak, işlev `correct\_classifications` sözlüğünü döndürür. Bu işlev, bir dosyadan sınıflandırma verilerini yüklemek ve doğrulamak için kullanışlıdır ve veri yapısının daha fazla işleme için beklendiği gibi olduğundan emin olur.

Sınıflandırmaları Güncelleme

```
1 def update_classifications(correct_classifications, blocks):
2     updated_classifications = []
3     for label, indices in correct_classifications.items():
4         for idx in indices:
5             if idx - 1 < len(blocks):
6                 updated_classifications.append({'code': blocks[idx - 1], 'classification': label})
7             else:
8                 print(f"Warning: Index {idx} is out of range for blocks list.")
9     return updated_classifications
```

`update\_classifications` işlevi, doğru sınıflandırmalarıyla bir kod bloğu listesini günceller. İki argüman alır: `correct\_classifications`, doğru etiketleri ve bunlara karşılık gelen dizinleri içeren bir sözlük ve `blocks`, kod blokları listesi.

Fonksiyon, güncellenmiş sınıflandırma sonuçlarını depolamak için boş bir `updated\_classifications` listesi başlatır. `correct\_classifications` sözlüğündeki her etiketi ve ilişkili dizinlerini inceler. Her dizin için, dizinin `blocks` listesinin aralığında olup olmadığını kontrol eder. Dizin geçerliyse, `updated\_classifications` listesine, verilen dizindeki kod bloğunu (sıfır tabanlı dizinleme için ayarlanmış) ve doğru etiketi içeren bir sözlük ekler. Dizin aralık dışındaysa, bir uyarı mesajı yazdırır.

Son olarak, fonksiyon, güncellenmiş sınıflandırmalarıyla birlikte kod bloklarını içeren `updated\_classifications` listesini döndürür.

Modeli Yeniden Eğitme

```
1 def retrain_model(data):
2     codes, classifications = preprocess_data(data)
3     X, vectorizer = vectorize_data(codes)
4     X_train, X_test, y_train, y_test = split_data(X, classifications)
5     y_train_encoded, label_to_index, index_to_label = label_encoding(y_train)
6     model = train_model(X_train, y_train_encoded)
7     save_model(model, vectorizer, label_to_index)
8     print("Model retrained and saved with updated classifications.")
9
10    print_training_info(X_train, X_test, y_train, y_test, model, index_to_label)
```

`retrain\_model` fonksiyonu, güncellenmiş verilerle bir makine öğrenimi modelini yeniden eğitir, güncellenmiş modeli ve bileşenlerini kaydeder ve eğitim bilgilerini yazdırır. Yeniden eğitim için kullanılacak veri kümesi olan `data` adlı bir argüman alır.

Fonksiyon, `preprocess\_data` fonksiyonunu kullanarak `kodları` ve `sınıflandırmaları` çıkarmak için verileri ön işleme tabi tutarak başlar. Daha sonra, `x` ve `vektörleştirici` özellik matrisini elde etmek için `vectorize\_data` fonksiyonunu kullanarak `kodları` vektörleştirir. Veriler, `split\_data` fonksiyonu kullanılarak eğitim ve test kümelerine ayrılır.

Daha sonra, eğitim etiketleri `label\_encoding` fonksiyonu kullanılarak sayısal değerlere kodlanır. Model, `train\_model` fonksiyonu kullanılarak eğitim verileri üzerinde eğitilir. Eğitimden sonra, model, vektörleştirici ve etiket-indeks eşlemesi `save\_model` fonksiyonu kullanılarak diske kaydedilir. Modelin yeniden eğitildiğini ve güncellenmiş sınıflandırmalarla kaydedildiğini belirten bir mesaj yazdırılır.

Son olarak, fonksiyon, model doğruluğu, bir sınıflandırma raporu ve bir karışıklık matrisi görselleştirmesi içeren `print\_training\_info` fonksiyonunu çağırarak ayrıntılı eğitim bilgilerini yazdırır.

Test ve Yeniden Eğitim

```
1 def test_and_retrain():
2     model, vectorizer, index_to_label = load_model()
3
4     test_code = read_test_code('test_code.py')
5     converted_code_blocks = convert_code_to_single_line_blocks(test_code)
6     filtered_blocks = [block for block in converted_code_blocks if not block.startswith('#')]
7     cleaned_blocks = remove_trailing_newlines(filtered_blocks)
8
9     classifications = classify_blocks(model, vectorizer, index_to_label, cleaned_blocks)
10
11     print("Classification results for test code:")
12     print_classification_results(classifications)
13
14     retrain_choice = input("Do you want to retrain the model with correct classifications? (yes/no): ").strip()
15     if retrain_choice.lower() == 'yes':
16         correct_file = input("Enter the filename containing correct classifications: ").strip()
17         correct_classifications = parse_classification_input(correct_file)
18
19         print("Correct Classifications:")
20         print(correct_classifications)
21         print(f"Type of correct classifications: {type(correct_classifications)}")
22
23         updated_classifications = update_classifications(correct_classifications, cleaned_blocks)
24
25         print("Newly classified data to be added:")
26         for entry in updated_classifications:
27             print(f"Code: {entry['code']}, Classification: {entry['classification']}")
28
29         original_data = load_data()
30         merged_data = merge_and_shuffle_data(original_data, updated_classifications)
31
32         save_data(merged_data)
33
34         retrain_model(merged_data)
35
36 if __name__ == "__main__":
37     test_and_retrain()
```

`test\_and\_retrain` fonksiyonu, eğitilmiş bir modeli yeni kod üzerinde test eder, isteğe bağlı olarak modeli doğru sınıflandırmalarla günceller ve modeli yeniden eğitir. Birkaç adımı tutarlı bir iş akışında birleştirir:

1. **Modeli Yükle :** Fonksiyon, `load\_model` fonksiyonunu kullanarak eğitilmiş modeli, vektörleştiriciyi ve etiket-indeks eşlemesini yükleyerek başlar.

2. **Test Kodunu Oku ve İşle :** `read\_test\_code` fonksiyonunu kullanarak bir dosyadan ('test_code.py') test kodunu okur. Daha sonra kod, `convert\_code\_to\_single\_line\_blocks` fonksiyonu kullanılarak tek satırlık bloklara dönüştürülür. Yorum karakteriyle (`#`) başlayan bloklar filtrelenebilir. Kalan bloklar, `remove\_trailing\_newlines` fonksiyonu kullanılarak sondaki yeni satırlardan temizlenir.

3. Kod Bloklarını Sınıflandır : Temizlenen bloklar, yüklenen modeli ve vektörleştiriciyi uygulayan `classify\_blocks` işlevi kullanılarak sınıflandırılır. Sınıflandırma sonuçları yazdırılır.

4. Yeniden Eğitim Seçeneği : Kullanıcıdan modeli doğru sınıflandırmalarla yeniden eğitip eğitmemeye karar vermesi istenir. Kullanıcı 'evet'i seçerse, işlev aşağıdaki adımlarla devam eder:

- **Doğru Sınıflandırmaları Girin :** Kullanıcıdan doğru sınıflandırmaları içeren dosya adını girmesi istenir. İşlev, `parse\_classification\_input` işlevini kullanarak bu dosyayı okur ve ayrıştırır.
- **Sınıflandırmaları Güncelleyin :** Doğru sınıflandırmalar yazdırılır ve işlev bunların türünü kontrol eder. İşlev daha sonra `update\_classifications` işlevini kullanarak kod bloklarının sınıflandırmalarını günceller. Yeni sınıflandırılan veriler yazdırılır.
- **Verileri Birleştir ve Kaydet :** İşlev, `load\_data` işlevini kullanarak orijinal verileri yükler. Orijinal veriler ve güncellenen sınıflandırmalar `merge\_and\_shuffle\_data` işlevi kullanılarak birleştirilir ve karıştırılır. Birleştirilen veriler `save\_data` işlevi kullanılarak kaydedilir.
- **Modeli Yeniden Eğit:** İşlev, `retrain\_model` işlevini çağırarak birleştirilen verileri kullanarak modeli yeniden eğitir.

5. Yürütme İşlevi: İşlev, komut dosyası doğrudan `if \_\_name\_\_ == "\_\_main\_\_":` kontrol edilerek çalıştırıldığında yürütülür.

Bu işlev, modeli test etme, güncelleme ve yeniden eğitim iş akışını entegre ederek, yeni verilere ve doğru sınıflandırmalara dayalı olarak makine öğrenimi modelinin sürekli iyileştirilmesine olanak tanır.

İlk Eğitim Sonuçları

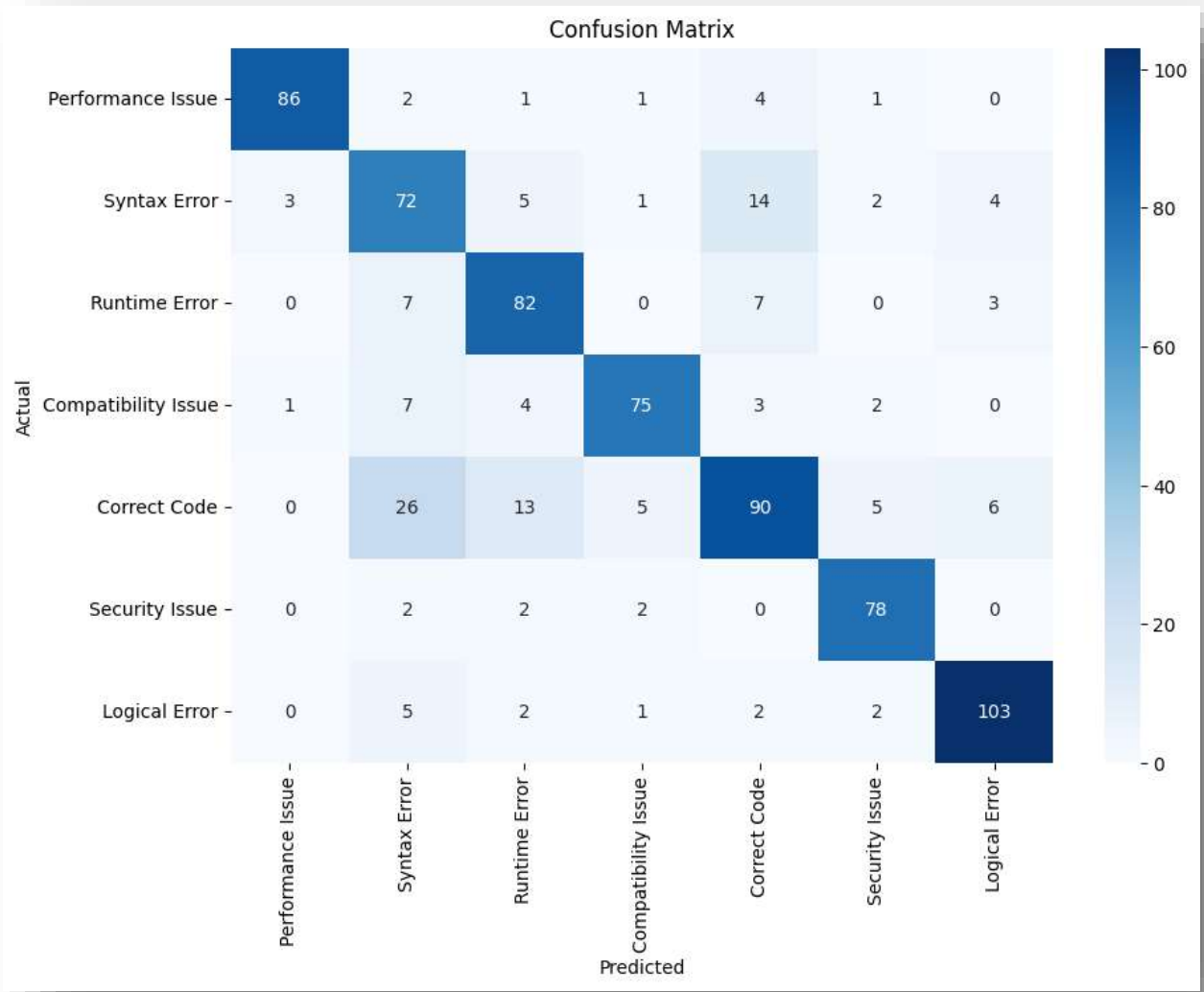
```
Loading data...
Data loaded.
Model, vectorizer, and label_to_index saved.
Model accuracy: 0.801641586867305
Classification report:
              precision    recall  f1-score   support

     0           0.96         0.91         0.93         95
     1           0.60         0.71         0.65        101
     2           0.75         0.83         0.79         99
     3           0.88         0.82         0.85         92
     4           0.75         0.62         0.68        145
     5           0.87         0.93         0.90         84
     6           0.89         0.90         0.89        115

 accuracy                   0.80         731
 macro avg           0.81         0.82         0.81         731
 weighted avg        0.81         0.80         0.80         731
```

Şekil 11 – İlk Model Eğitim Doğruluğu

Model başarıyla eğitildi ve kaydedildi. Modelin doğruluğu yaklaşık %80,16 idi. Sınıflandırma raporu her sınıf için ayrıntılı hassasiyet, geri çağırma ve f1 puanı ölçümleri sağlar.



Şekil 12 – İlk Model için Karmaşıklık Matrisi

Karmaşa matrisi, modelin performansını görsel olarak temsil eder, modelin her sınıfı ne kadar iyi tahmin ettiğini gösterir ve iyileştirme alanlarını vurgular

Test Verilerinin Örneği

```
1 # Function to find prime numbers up to n
2 def find_primes(n):
3     primes = []
4     for num in range(2, n + 1):
5         if all(num % i != 0 for i in range(2, int(num**0.5) + 1)):
6             primes.append(num)
7     return primes
8
9 print(find_primes(10000))
10
11 # Function with a syntax error
12 def syntax_error():
13     if True
14         print("This will not run")
15
16 syntax_error()
17
18 # Function to concatenate a list of strings
19 def concat_strings(strings):
20     result = ''
21     for s in strings:
22         result += s
23     return result
24
25 print(concat_strings(['a'] * 100000))
26
27 # Function to check if a string is a palindrome
28 def is_palindrome(s):
29     return s == s[::-1]
30
31 print(is_palindrome('radar'))
32
33 # Function to calculate the square root of a number
34 print(math.sqrt(-1))
35
36 # Class definition for a circle with area and circumference calculations
37 class Circle:
38     def __init__(self, radius):
39         self.radius = radius
40
41     def area(self):
42         return math.pi * self.radius ** 2
43
44     def circumference(self):
45         return 2 * math.pi * self.radius
46
47 circle = Circle(5)
48 print(circle.area())
49 print(circle.circumference())
```

Şekil 13 – Hatalı ve Doğru Kodlu Kod Örneği

Bu, sözdizimi hataları, performans sorunları ve mantıksal hatalar gibi çeşitli sorunları kapsayan hem doğru hem de yanlış kod örneklerini içeren test dosyasından basit bir örnektir.

İlk Modelle Test Sonuçları

```
1. LİBRARİLERİN YÜKLEME SÜRESİ (1000 ms)
2. Performans (1000 ms)
3. Syntax Error: 10 times in function [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]
4. Runtime Error: 10 times in function [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]
5. Compatibility Issue: 10 times in function [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]
6. Logical Error: 10 times in function [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]
7. Security Issue: 10 times in function [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]
8. Logical Error: 10 times in function [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150]
```

Şekil 14 – Test Kodu için Sınıflandırma Sonuçları

Test kodu sınıflandırıldı ve çeşitli sorun türleri ve belirli işlevler içindeki oluşumları belirlendi. Bu, kodun tam olarak nerede iyileştirilmesi gerektiğini belirlemeye yardımcı olur.

Doğru Sınıflandırma Görüntüsü

```
1 {
2   "Correct Code": [1,4,9,10,11,13,15,16,18,19,20,23,24,28,29,30,31,32,33,35,38,41,42,45,48,51,53,57,70,127,128,130],
3   "Security Issue": [5,8,39,40,46,58,66,81,94,103,113,116,125,131,134, 139, 144, 149, 154],
4   "Runtime Error": [2,3,7,17,32,34,36,44,52,55,56,60,63,65,67,69,72,78,79,80,83,86,89,92,99,100,101,102,105,108,111,114,115,117,118],
5   "Logical Error": [6,12,27,43,47,59,62,71,73,74,75,76,82,85,93,95,96,97,98,106,107,102,120,122],
6   "Syntax Error": [14,21,50,64,68,77,84,87,104,109,126,132, 137, 142, 147, 152, 157, 158],
7   "Compatibility Issue": [25,37,49,61,90,119,123,136, 141, 146, 151, 156],
8   "Performance Issue": [26,54,88,91,110,121,124,129,133, 138, 143, 148, 153]
9 }
```

Şekil 15 – Çeşitli Hata Türleri için Etiketli Veriler

Modelin bilgisini güncellemek ve doğruluğunu artırmak için doğru sınıflandırmalara sahip bir dosya kullanıldı.

Kod Dönüştürme ve Doğru Sınıflandırma Örneği

```
1 Code: import json\n\nclass DataProcessor:\n    def __init__(self, data):\n        self.data = data\n\n    def process_data(self):\n        processed_data = [d * 2 for d in self.data if d > 0]\n        return processed_data\n\n    def save_data(self, file_path):\n2 with open(file_path, 'w') as file:\n    json.dump(self.data, file)\n\ndata_processor = DataProcessor([1, 2, 3, -1, -2, -3])\nprint(data_processor.process_data())\n\ndata_processor.save_data('processed_data.json'), Classification: Correct Code\n3\n4 Code: def download_file(url, destination):\n    import urllib2\n    response = urllib2.urlopen(url)\n    with open(destination, 'wb') as out_file:\n        out_file.write(response.read())\n    return "Download complete"\n\nprint(download_file("http://example.com/file.txt", "file.txt")), Classification: Security Issue\n5\n6\n7 Code: def consonant_count(s):\n    consonants = 'bcdfghjklmnpqrstvwxyz'\n    return sum(1 for char in s if char.lower() not in consonants)\n\nprint(consonant_count('hello world')), Classification: Runtime Error\n8 Code: def read_text_file(file_path):\n    with open(file_path, 'r') as file:\n        return file.read()\n\nprint(read_text_file('nofile.txt')), Classification: Logical Error\n9 Code: def count_consonants_wrong(s):\n    consonants = 'bcdfghjklmnpqrstvwxyz'\n    return sum(1 for char in s if char.lower() not in consonants)\n\nprint(count_consonants_wrong('hello world')), Classification: Syntax Error\n10 Code: def read_csv_file(file_path):\n    import csv\n    with open(file_path, 'r') as file:\n        reader = csv.reader(file)\n        data = list(reader)\n        return data\n\nprint(read_csv_file('nonexistent_file.csv')), Classification: Compatibility Issue\n11 Code: def version_specific_code():\n    if sys.version_info[0] < 3:\n        print('Python 2.x')\n    else:\n        print('Python 3.x')\n\nversion_specific_code(), Classification: Performance Issue
```

Şekil 16 – Veri İşleme için Kod Örneği

Bu görüntü, test edilen kodun ve doğru sınıflandırmasının yeniden eğitim için nasıl düzenlendiğine dair bir örnek göstermektedir. Örnek, ham kodun ve sınıflandırmasının yapılandırılmış bir biçime dönüştürülmesini göstermektedir.

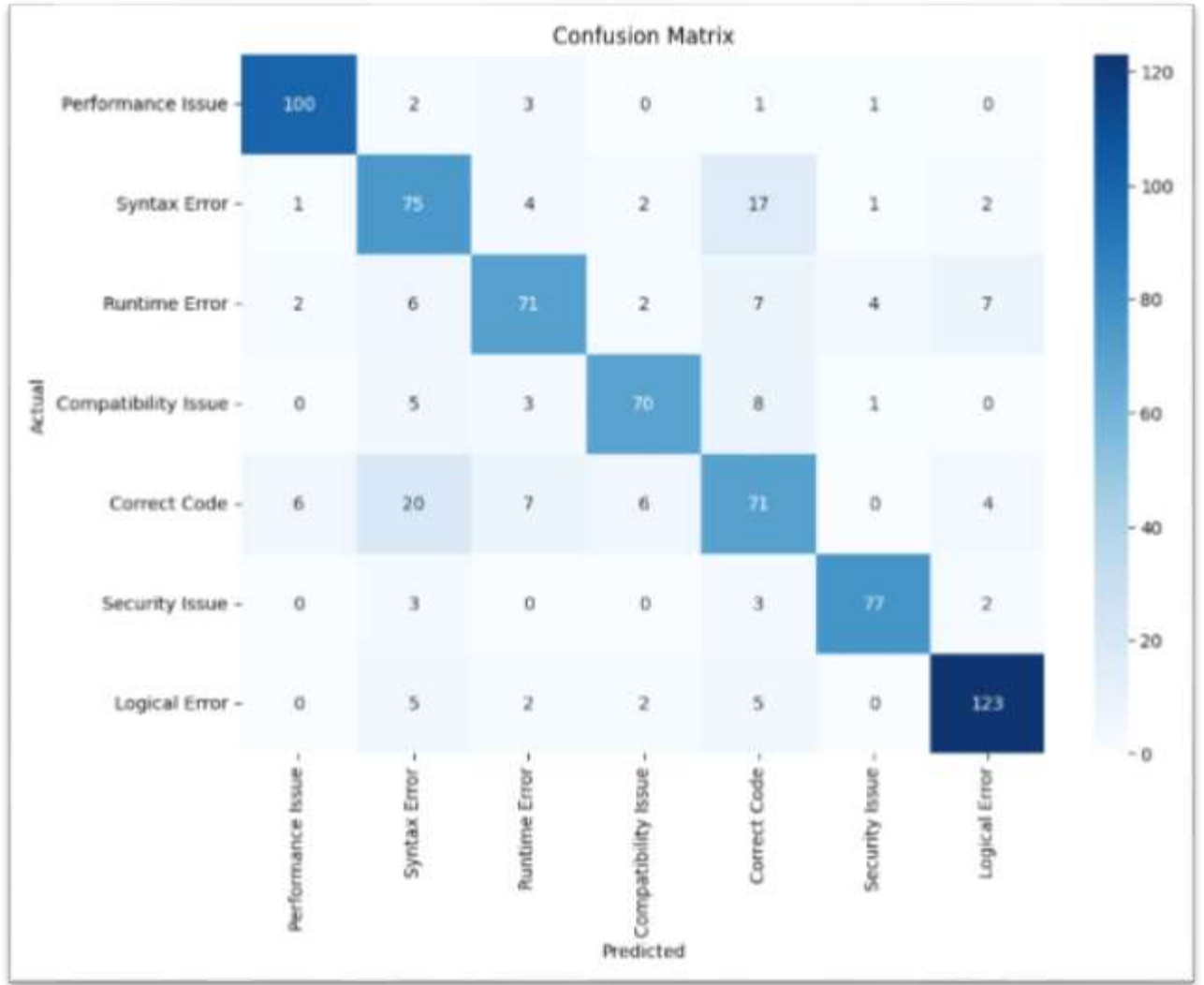
Yeniden Eğitilen Model Sonuçları

```
Updated data saved.  
Model, vectorizer, and label_to_index saved.  
Model retrained and saved with updated classifications.  
Model accuracy: 0.8030095759233926  
Classification report:
```

	precision	recall	f1-score	support
0	0.92	0.93	0.93	107
1	0.65	0.74	0.69	102
2	0.79	0.72	0.75	99
3	0.85	0.80	0.83	87
4	0.63	0.62	0.63	114
5	0.92	0.91	0.91	85
6	0.89	0.90	0.89	137
accuracy			0.80	731
macro avg	0.81	0.80	0.80	731
weighted avg	0.81	0.80	0.80	731

Şekil 17 – Yeniden Eğitilen Model Doğruluğu

Model, güncellenen sınıflandırmalar kullanılarak yeniden eğitildi. Doğruluk, yaklaşık %80,30'a kadar hafifçe iyileşti. Güncellenen sınıflandırma raporu, her sınıf için ayrıntılı performans ölçümlerini gösterir.



Şekil 18 – Yeniden Eğitilmiş Model için Karışıklık Matrisi

Güncellenen karışıklık matrisi, yeniden eğitilen modelin performansının görsel bir temsilini sağlar, iyileştirmeleri belirtir ve daha fazla iyileştirme için kalan alanları vurgular.

Bu sonuçlar, ilk veri toplama ve model eğitiminden test etmeye, sınıflandırmaları düzeltmeye ve doğruluğu artırmak için modeli yeniden eğitmeye kadar sürecin kapsamlı bir genel görünümünü sunmaktadır.

4.2.10. Aracın Özellikleri

Araç birkaç değerli özellik sunar:

- **Hata Sınıflandırması :** Farklı kod hatası türlerini belirler ve sınıflandırır.

- **Kod Önerileri** : Hataları düzeltmek için belirli öneriler sağlar.
- **Ayrıntılı Raporlar** : Kesinlik ve geri çağırma gibi ölçümleri içeren raporlar üretir.
- **Sürekli Öğrenme** : Doğruluğu artırmak için model yeni verilerle yeniden eğitilebilir.

4.2.11. Kısıtlamalar ve Sınırlamalar

Proje birkaç kısıtlamayla karşı karşıyadır :

- **Büyük Veri Gereksinimi** : Toplanması zor olan önemli miktarda etiketli veriye ihtiyaç duyar.
- **Kaynak Yoğun** : Modeli eğitmek ve çalıştırmak için önemli miktarda hesaplama kaynağı gerektirir.
- **Sınıflandırma Doğruluğu** : İlk model %100 doğru olmayabilir ve bazı hataları kaçırabilir veya yanlış sınıflandırabilir.

4.2.12. Sonuç ve Uygulamanın Sonuçları

Sonuçlar : İlk eğitim yaklaşık %80,16 doğruluk elde etti ve yeniden eğitilen model %80,30'a yükseldi.

Sonuç : Makine öğrenimi modeli, Python kod kalitesini sınıflandırmak ve iyileştirmek için etkili bir araçtır. Yeniden eğitim yoluyla sürekli iyileştirme, doğru ve alakalı kalmasını sağlar. Proje, makine öğreniminin yazılım geliştirmede, özellikle kod kalite güvencesinde pratik uygulamalarını göstermektedir.

5. TARTIŞMA

Çalışmamız, makine öğreniminin yazılım testinin çeşitli yönlerine nasıl yardımcı olabileceğini inceledi. Sunduğumuz örnekler, yapay zekanın yazılım test süreçlerini nasıl önemli ölçüde geliştirebileceğini gösteriyor. Başlangıçta, yapay zekanın testlerdeki potansiyelini geniş bir şekilde araştırdık ve başlangıçta tahmin edilenden daha fazla bilgi içeren, hızla gelişen bir alanı ortaya çıkardık. Bu tez, geleneksel test yöntemleri ile yeni ortaya çıkan AI teknikleri arasında köprü kurarak her iki alanda kullanılan terminoloji ve prosedürleri açıklamaktadır. AI araştırmasındaki geleneksel test kavramlarını bağlamsallaştırarak, makine öğrenimi yaklaşımlarının çeşitli test zorluklarını nasıl çözebileceğini gösterdik.

Araştırmamız, yazılım testinde yapay zekanın dönüştürücü potansiyelini vurguluyor. Alan geliştikçe, sürekli araştırma ve geliştirme şüphesiz daha da yenilikçi uygulamalara ve ilerlemelere yol açacaktır. Geleneksel ve AI test yöntemleri arasındaki bağlantı, makine öğreniminin uzun süredir devam eden test sorunlarını nasıl çözebileceğini ve sektörde potansiyel olarak devrim yaratabileceğini göstermesi açısından çok önemlidir.

Dahası, bu çalışma yapay zekanın yeteneklerini ve sınırlamalarını anlamının gerekliliğinin altını çiziyor. Yapay zeka çok sayıda fayda sunsa da, bunun her derde deva olmadığını anlamak önemlidir. Yapay zekanın yazılım testine entegrasyonu, hem geleneksel hem de AI-tabanlı test metodolojilerinin kapsamlı bir şekilde anlaşılmasını gerektirir. Bu anlayış, makine öğrenimi tekniklerinin etkili bir şekilde uygulanmasına olanak tanıyarak bunların insan uzmanlığının yerine geçmesini değil tamamlamasını sağlar.

Çalışmamızın uygulamalı kısmı, Python kodunun kalitesini otomatik olarak sınıflandırmak ve iyileştirmek için makine öğrenimi kullanan bir araç geliştirmeyi içeriyordu. Bu araç, geliştiricilere hataları vurgulayarak ve iyileştirmeler önererek anında geri bildirim sağlamak için test edildi ve gösterildi. Sonuçlar, Flake8 ve Pylint gibi araçların makine öğrenimi algoritmalarıyla entegre edilmesinin kod incelemelerinin verimliliğini ve doğruluğunu önemli ölçüde artırdığını gösterdi. Bu uygulamalı yaklaşım,

yazılım testinde AI kullanmanın pratik faydalarını ve zorluklarını göstererek bu alandaki daha fazla gelişme için bir yol haritası sunuyor.

Çalışma aynı zamanda yapay zekanın testlerde başarılı bir şekilde uygulanması için çeşitli kritik faktörleri de tanımlıyor. Bunlar, test ve geliştirme ekipleri içinde özel becerilere duyulan ihtiyacı ve alan geliştikçe sürekli öğrenmenin ve uyarlamanın önemini içerir. Araştırmamız, bu faktörleri vurgulayarak yazılım testlerinde yapay zekadan yararlanmak için kapsamlı bir çerçeve sağlıyor.



6. SONUÇ VE ÖNERİLER

Bu araştırma, yapay zekanın yazılım testini nasıl dönüştürebileceğini anlama yolunda önemli bir adıma işaret ediyor. Bu alan büyümeye devam ettikçe, devam eden araştırmalar ve yeni fikirlerin geliştirilmesi muhtemelen daha da çığır açıcı uygulamalara ve ilerlemelere yol açacaktır. Ancak araştırmamızın sınırlılıklarını kabul etmek önemlidir. Önerilen çözümlerin, gerçek-dünya senaryolarındaki etkinliklerine dair somut kanıtlar bulunmuyor ve testlerde makine öğrenimi yöntemlerini etkili bir şekilde kullanmak için özel becerilere ihtiyaç var.

Bu sınırlamalara rağmen bulgularımız yapay zekanın testlerde uygulanması için umut verici sonuçlar ortaya koyuyor. Yapay zeka, yazılım testinin verimliliğini ve doğruluğunu artırma, bu süreçler için gereken zamanı ve kaynakları azaltma potansiyeline sahiptir. Makine öğrenimi algoritmaları tekrarlanan görevleri otomatikleştirebilir, kalıpları belirleyebilir ve insan testçilerin gözden kaçırabileceği anormallikleri tespit edebilir. Ancak AI testinin verimlilik açısından geleneksel yöntemlere göre üstünlüğünü kanıtlayan kesin kanıtların bulunmaması gibi geçerli eleştirileri dikkate almak çok önemlidir. Ek olarak, belirli test senaryolarının karmaşıklığı, öngörülebilir gelecekte hala insan uzmanlığı gerektirebilir.

Bu araştırmanın uygulamalı yönü, Python kod kalitesini sınıflandırmak ve iyileştirmek için bir makine öğrenimi aracının geliştirilmesini ve uygulanmasını gösterdi. İlk eğitim yaklaşık %80,16 doğruluk elde etti ve yeniden eğitilen model %80,30'a yükseldi. Bu sonuçlar, makine öğrenimi modellerinin, yeniden eğitim yoluyla anında geri bildirim ve sürekli iyileştirme sağlayarak kod kalitesi güvencesi için etkili araçlar olabileceğini gösteriyor.

İleriye baktığımızda, AI testlerindeki ilerlemelerin üç temel adımı takip etmesi bekleniyor:

1. **Mevcut Araçların Geliştirilmesi:** Bu araştırmada incelenen araçların da gösterdiği gibi, makine öğrenimi yeteneklerini mevcut test araçlarına entegre etmek. Bu, verileri analiz edebilen, olası sorunları tahmin edebilen ve

iyileştirmeler önerebilen yapay zeka algoritmalarını içerecek şekilde mevcut yazılımın yükseltilmesini içerir. Bu araçları geliştirerek kuruluşlar, mevcut altyapılarını tamamen elden geçirmeden test süreçlerini iyileştirmek için yapay zekadan yararlanabilirler.

2. **Gelişmiş Çerçevelere Geçiş:** Geleneksel araçlardan daha gelişmiş AI test çerçevelerine geçiş yapılırken, makine öğrenimi çözümlerinin henüz mevcut olmadığı alanlarda insan uzmanlığına yönelik süregelen ihtiyacın kabul edilmesi. Bu geçiş, kuruluşların yavaş yavaş AI çerçevelerini benimsediği, ekiplerini eğittiği ve iş akışlarını bu yeni teknolojileri içerecek şekilde ayarladığı aşamalı bir yaklaşım gerektiriyor. Gelişmiş çerçeveler, otomatik test senaryosu oluşturma, akıllı test yürütme ve test sonuçlarından sürekli öğrenme dahil olmak üzere daha güçlü test yetenekleri sunabilir.
3. **Otonom Teste Doğru Gelişme:** Yapay zeka ile testlerin tamamen otomasyonu uzak bir olasılık olmasına rağmen, ilerlemeler sonunda insan katılımını en aza indirebilir. Otonom test, insan müdahalesi olmadan testleri bağımsız olarak tasarlayabilen, yürütebilen ve analiz edebilen AI sistemlerinin geliştirilmesini içerir. Bu düzeyde otomasyon henüz başlangıç aşamasındayken, devam eden araştırma ve geliştirme bizi bu gerçekliğe daha da yaklaştırabilir. Otonom testler insan hatasını önemli ölçüde azaltabilir, test hızını artırabilir ve daha tutarlı ve kapsamlı test kapsamı sağlayabilir.

Gelecekteki bu ilerlemelerin elde edilmesi, test uzmanları, mimarlar, geliştiriciler, veri bilimcileri ve yazılım geliştirme sürecindeki diğer paydaşlar dahil olmak üzere çeşitli uzmanlar arasında işbirliğini gerektirecektir. Bu işbirlikçi yaklaşım, yazılım testinde AI uygulamasının hem etkili hem de verimli olmasını sağlayacak ve sonuçta daha kaliteli yazılım ürünlerine yol açacaktır.

Bu iş birliğini kolaylaştırmak için kuruluşların, ekiplerini AI teknolojileriyle çalışmak için gerekli becerilerle donatan eğitim ve geliştirme programlarına yatırım yapmaları

gerekiyor. Farklı uzmanlıklara sahip üyelerden oluşan çapraz işlevli ekipler, inovasyonu ve problem çözmei teşvik ederek yapay zeka uygulamalarının, yazılım testinin benzersiz zorluklarına çözüm bulmasını sağlayabilir. Dahası, sürekli öğrenme ve deneme kültürünü teşvik etmek, kuruluşların çağın ilerisinde kalmasına ve hızla gelişen AI ortamına uyum sağlamasına yardımcı olabilir.

Sonuç olarak AI, yazılım testlerinde devrim yaratma konusunda büyük umut vaat ediyor olsa da, uygulamaya dengeli bir bakış açısıyla yaklaşmak çok önemli. Yazılım test endüstrisi, mevcut sınırlamaları kabul ederek ve bunları hedefe yönelik araştırma ve işbirliği yoluyla ele alarak yapay zekanın potansiyelinden tam anlamıyla yararlanabilir. Yazılım testinin geleceği, daha verimli, güvenilir ve ölçeklenebilir test süreçlerinin önünü açacak şekilde insan uzmanlığı ile makine zekasının kusursuz entegrasyonunda yatmaktadır.

KAYNAKLAR

Aggarwal, A., Shaikh, S., Hans, S., Haldar, S., Ananthanarayanan, R., & Saha, D. (2021). Testing Framework for Black-box AI Models. arXiv preprint arXiv:2102.06166v1 [cs.LG].

AGENT. (2019). AGENT. <https://bit.ly/3KcgIcd> (Accessed December 11, 2023).

Applitools, 2024. AI Powered Visual UI Testing and Monitoring. <https://bit.ly/4a76PHa> (Accessed January 13, 2024).

Apptest.ai. (2024). AI-Automated Testing. <https://bit.ly/44yAlUY> (Accessed April 1, 2024).

Authors of the Agile Manifesto (2001) The Agile Manifesto. Agile Alliance. <https://bit.ly/3wugLN8> (Accessed February 20, 2024).

Braga, Ronyerison & Neto, Pedro & Rabelo, Ricardo & Santiago, José & Souza, Matheus. (2018). A machine learning approach to generate test oracles. 142-151. 10.1145/3266237.3266273.

Božić, V. (2024). Intelligent Software Agents & Artificial Intelligence. DOI: 10.13140/RG.2.2.10380.08321.

Chen, J., Lou, Y., Zhang, L., Zhou, J., Wang, X., Hao, D., & Zhang, L. (2018). Optimizing test prioritization via test distribution analysis. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018). DOI: 10.1145/3236024.3236053.

Crispin, L. (2019, January 16). Practical Uses for Machine Learning in Software Delivery. <https://bit.ly/3JQYvAq> (Accessed February 1, 2024).

DrQA. (2019). Reading Wikipedia to Answer Open-Domain Questions. <https://bit.ly/3JZukHj> (Accessed March 1, 2024)

Dwarakanath, A., Ahuja, M., Sikand, S., Rao, R. M., Bose, R. P. J. C., Dubash, N., & Podder, S. (2018). Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018) (pp. 118–128). DOI: 10.1145/3213846.3213858.

Diffblue. (2024). Diffblue - AI for Code. <https://bit.ly/3WxZltG> (Accessed March 15, 2024).

Eric , T. (2019). Mabl - The truth about ML in test automation. <https://bit.ly/3WzFxpI> (Accessed February 20, 2024)

Figure Eight. (2018). Data Scientist Report 2018. <https://bit.ly/4acKhVe> (Accessed November 10, 2023)

Forrester. (2023). The Forrester Wave: Robotic Process Automation Suites, Q4 2023. <https://bit.ly/3WxZiOw> (Accessed February 29, 2024)

Functionize. (2024). Automation Testing with Machine Learning. <https://bit.ly/4ahpmAu> (Accessed January 11, 2024)

George, A. S., George, A. H., & Baskar, T. (2023). The Rise of Hyperautomation: A New Frontier for Business Process Automation. <https://bit.ly/4blkRpE> (Accessed February 29, 2024)

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (2nd ed.). O'Reilly Media.

Gerrard, P. (2017). The New Model for Testing. <https://bit.ly/3wvxIqv> (Accessed November 20, 2023)

Gerrard, P. (2020). The Tester Skills Program: Teaching Testers to Think for Themselves. In S. Goericke (Ed.), The Future of Software Quality Assurance (pp. 1-12). Gerrard Consulting, Macclesfield, UK. https://doi.org/10.1007/978-3-030-29509-7_4

Harvard Business Review. (2022). The Business Case for Explainable AI. <https://bit.ly/3QCm1VT> (Accessed February 29, 2024)

Ho, T.K. (1995). Random Decision Forest. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, 14-16 August 1995, 278-282.

Hristov, A. (2019, June). End-to-End Testing: Revolutionized. <https://bit.ly/3WxTzbo> (Accessed December 10, 2023)

Hu, Gang & Zhu, Linjie & Yang, Junfeng. (2018). AppFlow: using ML to synthesize robust. 269-282. 10.1145/3236024.3236055.

InfoQ. (2023). Software Testing, Artificial Intelligence and Machine Learning Trends in 2023. <https://bit.ly/3QCYlk7> (Accessed February 29, 2024)

ISTQB. (2018). ISTQB Certified Tester Foundation Level - Syllabus 2018. International Software Testing Qualifications Board. <https://bit.ly/3UT50rV> (Accessed February 29, 2024)

ISTQB. (2023). ISTQB Certified Tester Foundation Level - Syllabus 2023. International Software Testing Qualifications Board. <https://bit.ly/3JRSnrJ> (Accessed February 29, 2024)

Jason A, Navrides, C., Toley, K., Bedino, R., Fan, V., & Petersen, M. (2018). Abstract Intent Test (AIT) Syntax. <https://bit.ly/3yalJPw> (Accessed December 29, 2023)

Johnson, R. C. (2018). Overcoming AI Bias with AI Fairness. <https://bit.ly/3UUtf99> (Accessed February 23, 2024).

Keysight. (2024). Software Testing. <https://bit.ly/3QCjdrM> (Accessed January 15, 2024).

Khokhar, M. N., Bashir, M. B., & Fiaz, M. (2020). Metamorphic Testing of AI-based Applications: A Critical Review. International Journal of Advanced Computer Science and Applications, 11(4).

Khyati ,M. (2020). The Powerful Bayes' Theorem for Data Science Professionals. <https://bit.ly/4bv0Exz> (Accessed February 20, 2024)

LambdaTest. (2023). Future of Quality Assurance 2023. <https://bit.ly/44B4CTl> (Accessed February 29, 2024)

Linux Foundation. (2021). Cloud Native Performance Testing. <https://bit.ly/3WxZpcX> (Accessed February 29, 2024)

Lomas, N. (2019, August 6). TechCrunch. <https://tcrn.ch/4adnJ75> (Accessed February 13, 2024)

Lima, R., Rosado da Cruz, A. M., & Ribeiro, J. (2020). Artificial Intelligence Applied to Software Testing: A Literature Review. In 2020 15th Iberian Conference on Information Systems and Technologies (CISTI). DOI: 10.23919/CISTI49556.2020.9141124.

Lisa Crispin, & Janet Gregory. (2009). Agile Testing: A Practical Guide for Testers and Agile Teams (1st ed.). Addison-Wesley.

Luo, Qi; Poshyvanyk, Denys; Nair, Aswathy; and Grechanik, Mark. (2016). tool for detecting performance problems with feedback-driven learning software testing. 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). 10.1145/2889160.2889164

Marijan, Dusica & Gotlieb, Arnaud & Ahuja, Mohit. (2019). Challenges of Testing Machine Learning Based Systems. 101-102. 10.1109/AITest.2019.00010.

Martinez, Jorge & Thomas, Troy & Tariq. (2014). middleware architecture for domain-specific UI test automation. 10.1145/2631890.2631893.

Meinke, Karl & Bennaceur, Amel. (2017). Machine Learning for Software Engineering Models, Methods, and Applications. 10.1145/3183440.3183461.

Meinke, Karl & Niu, Fei & Sindhu, Muddassar. (2011). Learning-Based Software Testing: A Tutorial. Communications in Computer and Information Science. 336. 10.1007/978-3-642-34781-8_16.

Memon, Atif & Pollack, M.E. & Soffa, Mary. (1999). Generate test cases for GUIs. 257-266. 10.1109/ICSE.1999.841016.

Melisha. (2018, October 21). How AI can improve pentesting. <https://bit.ly/3WzuYDg> (Accessed March 20, 2024)

Microsoft Azure. (2023). Azure DevOps Services: Accelerate your software delivery pipeline. <https://bit.ly/4bpUTRM> (Accessed February 29, 2024)

MIT Technology Review. (2023). How AI is making software testing smarter. <https://bit.ly/4bv0FBD> (Accessed February 29, 2024)

Daniel J. M., & Bruce A. P. (2002). Software Test Automation. (Just Enough (Yourdon Press))

Myers, G. J., Sandler, C., & Badgett, T. (2011). The Art of Software Testing (3rd ed.). Wiley.

OWASP (Open Web Application Security Project). (2020). Testing Guide. <https://bit.ly/4brebpZ> (Accessed February 29, 2024)

perfecto.io. (2024). Perfecto Scriptless Mobile Automate the Complex, No Matter the Scale. <https://bit.ly/3QHc0GG> (Accessed April 15, 2024)

Reid, S. (2014). ISO/IEC/IEEE 29119: The New International Software Testing Standards. <https://bit.ly/3QB009M> (Accessed February 29, 2024)

Remo, L. (2018). Machine Learning-Driven Test Case Prioritization Approaches for Black-Box Testing. In Proceedings of ettc2018 - European Test and Telemetry Conference. DOI: 10.5162/ettc2018/12.4.

Richard, S., & Andrew, G. (2018). Reinforcement learning: An introduction (2nd ed.). The MIT Press.

Rodriguez, Daniel & Dolado, Jose & Tuya, Javier. (2015). Bayesian concepts in software testing: an initial review. 41-46. 10.1145/2804322.2804329.

Roper, Marc. (2019). Using Machine Learning to Classify Test Outcomes. 99-100. 10.1109/AITest.2019.00009.

Santiago, Dionny & Clarke, Peter & Alt, Patrick & Tariq. (2018). Abstract flow learning for web application test generation. 49-55. 10.1145/3278186.3278194.

PENTOMA. (2018, February 27). PENTOMA The New AI-Powered Pen Testing Solution. <https://bit.ly/3UQq3MF> (Accessed March 12, 2024)

Siraj, R. (2017). Q Learning Explained (tutorial). <https://bit.ly/4bcAFLQ> (Accessed January 29, 2024)

Sofy. (2024). App testing |Sofy.ai. <https://bit.ly/4dL8PIj> (Accessed April 1, 2024)

Splunk. (2024). Artificial Intelligence | Splunk AI. <https://bit.ly/44AD00v> (Accessed March 11, 2024)

Spieker, Helge & Gotlieb, Arnaud & Marijan, Dusica & Mossige, Morten. (2017). Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration. 10.1145/3092703.3092709.

Stuart ,R & Peter ,N. (2016). Artificial Intelligence: A Modern Approach (3rd ed.). Pearson Education Limited.

Testim.io. (2024). Move faster and reduce costs with the power of AI. <https://bit.ly/3WxZLjN> (Accessed February 24, 2024)

Testrigor. (2024). Generative AI-based Test Automation Tool. <https://bit.ly/44Blk4S> (Accessed April 1, 2024).

Tariq M. (2018). Artificial intelligence and machine learning for the Testing World. <https://bit.ly/3UV1puh> (Accessed November 11, 2023).

Tariq M., Jason, A., Santiago, D., Adamo, D., Chin, W., & Shanmugam, R. (2019). Artificial Intelligence for Testing Today and Tomorrow: Industry Perspectives. In 2019 IEEE International Conference On Artificial Intelligence Testing (AITest). IEEE. DOI: 10.1109/AITest.2019.000-3

University of Helsinki & Reaktor. (2024). Elements of AI. <https://bit.ly/4bmtksQ> (Accessed March 20, 2024)

World Quality Report (2022). <https://bit.ly/4btWVAj> (Accessed February 29, 2024)

Zhang, J., Wang, Z., Zhang, L., Hao, D., Zang, L., Cheng, S., & Zhang, L. (2016). Predictive mutation testing. In Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016), (pp. 342–353). DOI: 10.1145/2931037.2931038.

EKLER

EK 1. Veri Toplama, Analiz ve Hata Sınıflandırması için Kod :

```
import os
import subprocess
import json
import tempfile
import concurrent.futures
import time
from stackapi import StackAPI
from bs4 import BeautifulSoup
import random

# List of repositories and directories
REPOS = [
    ('TheAlgorithms/Python', 'Python-master'),
    ('geekcomputers/Python', 'Python-master (1)'),
    ('leachim6/hello-world', 'hello-world-main'),
    ('labuladong/fucking-algorithm', 'algorithm-master'),
    ('python/cpython', 'cpython-main'),
]

def collect_local_code(repo_dir):
    codes = []
    for root, _, files in os.walk(repo_dir):
        for file in files:
            if file.endswith('.py'):
                codes.append(os.path.join(root, file))
    return codes

def analyze_code_with_pylint(file_path):
    result = subprocess.run(['pylint', file_path], stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    pylint_output = result.stdout.decode('utf-8') + result.stderr.decode('utf-8')
    return pylint_output

def analyze_code_with_flake8(file_path):
    result = subprocess.run(['flake8', file_path], stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    flake8_output = result.stdout.decode('utf-8')
    return flake8_output

def classify_error(output):
```

```

if "syntax-error" in output:
    return "Syntax Error"
elif "runtime-error" in output:
    return "Runtime Error"
elif "logical-error" in output:
    return "Logical Error"
elif "performance" in output:
    return "Performance Issue"
elif "compatibility" in output:
    return "Compatibility Issue"
elif "security" in output:
    return "Security Issue"
else:
    return "Correct Code"

def process_repository(repo):
    repo_name, repo_dir = repo
    code_files = collect_local_code(repo_dir)
    repo_analysis = []

    for file_path in code_files:
        try:
            with open(file_path, 'r', encoding='utf-8') as code_file:
                code = code_file.read()

            pylint_output = analyze_code_with_pylint(file_path)
            flake8_output = analyze_code_with_flake8(file_path)

            classification = classify_error(pylint_output + flake8_output)

            repo_analysis.append({
                'code': code,
                'classification': classification
            })

        except Exception as e:
            repo_analysis.append({
                'code': code,
                'classification': 'Error'
            })

    return repo_analysis

start_time = time.time()

all_codes = []
with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:

```

```

future_to_repo = {executor.submit(process_repository, repo): repo for repo in
REPOS}
for future in concurrent.futures.as_completed(future_to_repo):
    try:
        repo_analysis = future.result()
        all_codes.extend(repo_analysis)
    except Exception as e:
        print(f"Error processing repository: {future_to_repo[future]}, error: {str(e)}")

# Generate complex code samples
def generate_syntax_error():
    syntax_errors = [
        "def read_file(file_path):\nwith open(file_path, 'r') as file:\n    content =
file.read()", # Missing colon after function definition
        "class MyClass:\n    def __init__(self, value)\n        self.value = value", # Missing
colon after __init__
        "def fetch_data(url):\nresponse = requests.get(url)\nreturn response.json()", #
Missing colon after function definition
        "if x > 10\nprint('x is greater than 10')", # Missing colon after if
    ]
    return random.choice(syntax_errors)

def generate_runtime_error():
    runtime_errors = [
        "def read_file(file_path):\n    with open(file_path, 'r') as file:\n        return
file.read()\nread_file('nonexistent_file.txt')", # Attempting to open a nonexistent file
        "class MyClass:\n    def __init__(self, value):\n        self.value = value\nobj =
MyClass()\nprint(obj.value)", # Missing argument in constructor
        "import requests\nresponse =
requests.get('http://nonexistent.url')\nprint(response.text)", # Attempting to access a
nonexistent URL
    ]
    return random.choice(runtime_errors)

def generate_logical_error():
    logical_errors = [
        "def read_file(file_path):\n    with open(file_path, 'r') as file:\n        return
file.read()\ncontent = read_file('file.txt')\nif content:\n    print('File is empty')", #
Incorrect logic for checking file content
        "class MyClass:\n    def __init__(self, value):\n        self.value = value\n    def
is_positive(self):\n        return self.value < 0\nobj =
MyClass(10)\nprint(obj.is_positive())", # Incorrect logic for checking positive value
        "import requests\nresponse = requests.get('http://example.com')\nif
response.status_code == 200:\n    print('Failed to fetch data')", # Incorrect logic for
checking response status
    ]
    return random.choice(logical_errors)

```

```

def generate_performance_issue():
    performance_issues = [
        "def read_large_file(file_path):\n    with open(file_path, 'r') as file:\n        content =\nfile.read()\n    return content\nread_large_file('large_file.txt')", # Loading the entire file
into memory
        "class MyClass:\n    def __init__(self, values):\n        self.values = values\n    def\ncalculate_sum(self):\n        result = 0\n        for value in self.values:\n            result +=\nvalue\n        return result\nobj =\nMyClass(range(1000000))\nprint(obj.calculate_sum())", # Inefficient sum calculation
        "import requests\nurls = ['http://example.com'] * 1000\nresponses =\n[requests.get(url) for url in urls]\nprint(len(responses))", # Sending synchronous
requests instead of using an asynchronous library
    ]
    return random.choice(performance_issues)

def generate_compatibility_issue():
    compatibility_issues = [
        "print 'Hello, world!'", # Using Python 2 print statement
        "xrange(10)", # Using Python 2 xrange
        "def handle_bytes(data):\n    if isinstance(data, str):\n        data = data.encode('utf-\n8')\n    return data\nhandle_bytes(u'unicode string')", # Compatibility issue with string
handling
        "import configparser\nconfig =\nconfigparser.ConfigParser()\nconfig.read('config.ini')", # Using an old library
potentially incompatible with modern versions
    ]
    return random.choice(compatibility_issues)

def generate_security_issue():
    security_issues = [
        "def insecure_eval(user_input): return eval(user_input)", # Using eval
        "def insecure_exec(user_input): exec(user_input)", # Using exec
        "def sql_injection(user_input):\n    query = \"SELECT * FROM users WHERE\nname = '\" + user_input + '\"\"\n    execute_query(query)", # Vulnerable to SQL
injection
        "def insecure_pickle_load(data):\n    import pickle\n    return pickle.loads(data)", #
Insecure deserialization with pickle
        "def insecure_os_system(command):\n    import os\n    os.system(command)", #
Insecure use of os.system
    ]
    return random.choice(security_issues)

def generate_correct_code():
    correct_codes = [
        "def read_file(file_path):\n    with open(file_path, 'r') as file:\n        return\nfile.read()",
        "class MyClass:\n    def __init__(self, value):\n        self.value = value\n    def\nget_value(self):\n        return self.value",
    ]

```

```

        "import requests\nresponse = requests.get('http://example.com')\nif
response.status_code == 200:\n    print('Data fetched successfully')",
        "def process_data(data):\n    processed = [d * 2 for d in data]\n    return processed",
        "class Animal:\n    def __init__(self, name):\n        self.name = name\n    def
speak(self):\n        raise NotImplementedError('Subclasses must implement this
method')\n\nclass Dog(Animal):\n    def speak(self):\n        return 'Woof!'",
    ]
    return random.choice(correct_codes)

# Generate complex code samples
def generate_complex_code_samples(num_samples):
    samples = []
    for _ in range(num_samples):
        error_type = random.choice([
            "Syntax Error", "Runtime Error", "Logical Error",
            "Performance Issue", "Compatibility Issue", "Security Issue", "Correct Code"
        ])
        if error_type == "Syntax Error":
            code = generate_syntax_error()
        elif error_type == "Runtime Error":
            code = generate_runtime_error()
        elif error_type == "Logical Error":
            code = generate_logical_error()
        elif error_type == "Performance Issue":
            code = generate_performance_issue()
        elif error_type == "Compatibility Issue":
            code = generate_compatibility_issue()
        elif error_type == "Security Issue":
            code = generate_security_issue()
        elif error_type == "Correct Code":
            code = generate_correct_code()

        samples.append({
            'code': code,
            'classification': error_type
        })

    return samples

# Generate 1000 complex examples
complex_code_samples = generate_complex_code_samples(1000)
all_codes.extend(complex_code_samples)

print("Generated complex code samples.")

# Collect StackOverflow data
SITE = StackAPI('stackoverflow')
SITE.max_pages = 25

```

```

SITE.page_size = 100

def extract_code(html_content):
    soup = BeautifulSoup(html_content, 'html.parser')
    code_snippets = [code.get_text() for code in soup.find_all('code')]
    return "\n".join(code_snippets)

def collect_stackoverflow_data(start_date, end_date):
    questions = []
    for page in range(1, SITE.max_pages + 1):
        response = SITE.fetch('questions', tagged='python', filter='withbody',
sort='creation', order='desc', fromdate=start_date, todate=end_date, page=page)
        questions.extend(response['items'])
        if len(response['items']) < SITE.page_size:
            break
        time.sleep(2)

    code_snippets = []
    for question in questions:
        if 'code' in question['body']:
            code_content = extract_code(question['body'])
            try:
                with tempfile.NamedTemporaryFile(delete=False, suffix=".py") as temp_file:
                    temp_file.write(code_content.encode('utf-8'))
                    temp_file_path = temp_file.name

                pylint_output = analyze_code_with_pylint(temp_file_path)
                flake8_output = analyze_code_with_flake8(temp_file_path)

                classification = classify_error(pylint_output + flake8_output)
                os.remove(temp_file_path)
            except Exception as e:
                classification = 'Error'

            code_snippets.append({
                'code': code_content,
                'classification': classification
            })

    return code_snippets

date_ranges = [
    (1609459200, 1609545600),
    (1609545600, 1609632000),
]

for start_date, end_date in date_ranges:
    stackoverflow_data = collect_stackoverflow_data(start_date, end_date)

```

```

all_codes.extend(stackoverflow_data)
time.sleep(10)

# Clean the collected data
cleaned_code_data = [entry for entry in all_codes if entry['code'].strip()]

with open("cleaned_training_data_Final.json", "w", encoding="utf-8") as f:
    json.dump(cleaned_code_data, f, indent=4)

end_time = time.time()
execution_time = end_time - start_time
print(f>Data collection and cleaning complete. Cleaned data saved to
cleaned_training_data_Final.json")
print(f>Total execution time: {execution_time} seconds")

```

EK 2. Kod Sınıflandırması için Makine Öğrenme Modelini Eğitme ve Değerlendirme Kodu :

```

import json
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import xgboost as xgb
import joblib
import matplotlib.pyplot as plt
import seaborn as sns

def load_data():
    print("Loading data...")
    with open('cleaned_training_data_Final.json', 'r', encoding='utf-8') as f:
        data = json.load(f)
    print("Data loaded.")
    return data

def preprocess_data(data):
    codes = [entry['code'] for entry in data]
    classifications = [entry['classification'] for entry in data]
    return codes, classifications

def vectorize_data(codes):
    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(codes)
    return X, vectorizer

```

```

def split_data(X, y):
    return train_test_split(X, y, test_size=0.2, random_state=42)

def label_encoding(y):
    label_to_index = {label: index for index, label in enumerate(set(y))}
    index_to_label = {index: label for label, index in label_to_index.items()}
    y_encoded = [label_to_index[label] for label in y]
    return y_encoded, label_to_index, index_to_label

def train_model(X_train, y_train):
    model = xgb.XGBClassifier(eval_metric='mlogloss')
    model.fit(X_train, y_train)
    return model

def save_model(model, vectorizer, label_to_index):
    joblib.dump(model, 'xgb_model.pkl')
    joblib.dump(vectorizer, 'vectorizer.pkl')
    joblib.dump(label_to_index, 'label_to_index.pkl')
    print("Model, vectorizer, and label_to_index saved.")

def print_training_info(X_train, X_test, y_train, y_test, model, index_to_label):
    y_test_encoded, _, _ = label_encoding(y_test)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test_encoded, y_pred)
    print(f"Model accuracy: {accuracy}")
    print("Classification report:")
    print(classification_report(y_test_encoded, y_pred))

    cm = confusion_matrix(y_test_encoded, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=index_to_label.values(), yticklabels=index_to_label.values())
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

def main():
    data = load_data()
    codes, classifications = preprocess_data(data)
    X, vectorizer = vectorize_data(codes)
    X_train, X_test, y_train, y_test = split_data(X, classifications)
    y_train_encoded, label_to_index, index_to_label = label_encoding(y_train)
    model = train_model(X_train, y_train_encoded)

```

```

save_model(model, vectorizer, label_to_index)
print_training_info(X_train, X_test, y_train, y_test, model, index_to_label)

if __name__ == "__main__":
    main()

```

EK 3. Python Kod Bloklarını Yükleme, Test Etme ve Sınıflandırma Kodu :

```

import json
import joblib

def load_model():
    model = joblib.load('xgb_model.pkl')
    vectorizer = joblib.load('vectorizer.pkl')
    label_to_index = joblib.load('label_to_index.pkl')
    index_to_label = {index: label for label, index in label_to_index.items()}
    return model, vectorizer, index_to_label

def load_data(file_path='cleaned_training_data_Final.json'):
    with open(file_path, 'r', encoding='utf-8') as f:
        data = json.load(f)
    return data

```

```

def read_test_code(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        return f.read()

def convert_code_to_single_line_blocks(code):
    lines = code.split('\n')
    blocks = []
    current_block = []
    inside_block = False

    for line in lines:
        stripped_line = line.rstrip()
        if stripped_line.startswith('#'):
            if current_block:
                blocks.append("\n".join(current_block))
                current_block = []
            blocks.append(stripped_line)
            inside_block = False
        else:
            current_block.append(stripped_line)
            inside_block = True

```

```

if current_block:
    blocks.append("\n".join(current_block))

return blocks

def remove_trailing_newlines(blocks):
    cleaned_blocks = []
    for block in blocks:
        cleaned_block = block.rstrip("\n")
        cleaned_blocks.append(cleaned_block)
    return cleaned_blocks

def classify_blocks(model, vectorizer, index_to_label, blocks):
    classifications = []
    for block in blocks:
        features = vectorizer.transform([block])
        prediction_index = model.predict(features)
        prediction_label = index_to_label[prediction_index[0]]
        classifications.append(prediction_label)
    return classifications

def print_classification_results(classifications):
    result_counts = {label: {'count': 0, 'indices': []} for label in set(classifications)}
    for i, label in enumerate(classifications, 1):
        result_counts[label]['count'] += 1
        result_counts[label]['indices'].append(i)

    for label, info in result_counts.items():
        print(f"{label}: {info['count']} times in functions {info['indices']}")

```

EK 4. Sınıflandırmaları Güncelleme ve Modeli Yeniden Eğitim Kodu :

```

import random

def parse_classification_input(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        correct_classifications = json.load(f)
    return correct_classifications

def update_classifications(correct_classifications, blocks):
    updated_classifications = []
    for label, indices in correct_classifications.items():
        for idx in indices:
            updated_classifications.append({'code': blocks[idx - 1], 'classification': label})
    return updated_classifications

```

```

def merge_and_shuffle_data(original_data, new_data):
    codes_in_original = {entry['code'] for entry in original_data}
    merged_data = original_data[:]
    for entry in new_data:
        if entry['code'] not in codes_in_original:
            merged_data.append(entry)
    random.shuffle(merged_data)
    return merged_data

def save_data(data, file_path='cleaned_training_data_Final.json'):
    with open(file_path, 'w', encoding='utf-8') as f:
        json.dump(data, f, ensure_ascii=False, indent=4)
    print("Updated data saved.")

def retrain_model(data):
    codes, classifications = preprocess_data(data)
    X, vectorizer = vectorize_data(codes)
    X_train, X_test, y_train, y_test = split_data(X, classifications)
    y_train_encoded, label_to_index, index_to_label = label_encoding(y_train)
    model = train_model(X_train, y_train_encoded)
    save_model(model, vectorizer, label_to_index)
    print("Model retrained and saved with updated classifications.")

def parse_classification_input(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        correct_classifications = json.load(f)

    if not isinstance(correct_classifications, dict):
        raise ValueError("Correct classifications should be a dictionary.")

    return correct_classifications

def update_classifications(correct_classifications, blocks):
    updated_classifications = []
    for label, indices in correct_classifications.items():
        for idx in indices:
            if idx - 1 < len(blocks):
                updated_classifications.append({'code': blocks[idx - 1], 'classification': label})
            else:
                print(f"Warning: Index {idx} is out of range for blocks list.")
    return updated_classifications

def print_training_info(X_train, X_test, y_train, y_test, model, index_to_label):
    y_test_encoded, _, _ = label_encoding(y_test)
    y_pred = model.predict(X_test)

```

```

accuracy = accuracy_score(y_test_encoded, y_pred)
print(f"Model accuracy: {accuracy}")
print("Classification report:")
print(classification_report(y_test_encoded, y_pred))

cm = confusion_matrix(y_test_encoded, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=index_to_label.values(), yticklabels=index_to_label.values())
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

def retrain_model(data):
    codes, classifications = preprocess_data(data)
    X, vectorizer = vectorize_data(codes)
    X_train, X_test, y_train, y_test = split_data(X, classifications)
    y_train_encoded, label_to_index, index_to_label = label_encoding(y_train)
    model = train_model(X_train, y_train_encoded)
    save_model(model, vectorizer, label_to_index)
    print("Model retrained and saved with updated classifications.")

    print_training_info(X_train, X_test, y_train, y_test, model, index_to_label)

def test_and_retrain():
    model, vectorizer, index_to_label = load_model()

    test_code = read_test_code('test_code.py')
    converted_code_blocks = convert_code_to_single_line_blocks(test_code)
    filtered_blocks = [block for block in converted_code_blocks if not
block.startswith('#')]
    cleaned_blocks = remove_trailing_newlines(filtered_blocks)

    classifications = classify_blocks(model, vectorizer, index_to_label, cleaned_blocks)

    print("Classification results for test code:")
    print_classification_results(classifications)

    retrain_choice = input("Do you want to retrain the model with correct classifications?
(yes/no): ").strip()
    if retrain_choice.lower() == 'yes':
        correct_file = input("Enter the filename containing correct classifications: ").strip()
        correct_classifications = parse_classification_input(correct_file)

        print("Correct Classifications:")
        print(correct_classifications)
        print(f"Type of correct classifications: {type(correct_classifications)}")

```

```
updated_classifications = update_classifications(correct_classifications,
cleaned_blocks)

print("Newly classified data to be added:")
for entry in updated_classifications:
    print(f"Code: {entry['code']}, Classification: {entry['classification']}")

original_data = load_data()
merged_data = merge_and_shuffle_data(original_data, updated_classifications)

save_data(merged_data)

retrain_model(merged_data)

if __name__ == "__main__":
    test_and_retrain()
```

