STS: Yapay Zeka Destekli Akıllı Test Senaryosu Oluşturma Aracı

STS: AI-Driven Smart Test Scenario Generation Tool

Cem Bağlum
Yönetim Bilişim Sistemleri
Fenerbahçe Üniversitesi
İstanbul, Türkiye
cem.baglum@fbu.edu.tr

Uğur Yayan
Yazılım Mühendisliği
Eskişehir Osmangazi Üniversitesi
Akıllı Sistemler Uygulama ve Araştırma Merkezi
Otonom Sistemler ve Güvenilirlik Laboratuvarı (ESOGÜ - ASRLab)
Eskişehir, Türkiye
ugur.yayan@ogu.edu.tr

***Özetçe*—Yazılım test yaşam döngüsünün en kritik aşamalarından biri olan test senaryosu oluşturma süreci, yüksek zaman ve kaynak gereksinimleri nedeniyle test faaliyetlerinin genel verimliliğini olumsuz etkilemektedir. Bu soruna yenilikçi bir yaklaşım olarak geliştirilen Akıllı Test Senaryosu Aracı (STS), büyük dil modellerini kullanarak xlsx, py, cpp, txt ve docx formatlarındaki dokümanları analiz eder ve bu analizler doğrultusunda bağlamsal olarak tutarlı, otomatik test senaryoları üretir. Bu yöntem, manuel test süreçlerinde sıklıkla karşılaşılan zaman kaybı ve insan hatası riskini en aza indirerek, test süreçlerini sistematik ve bağlama duyarlı hale getirmektedir. Streamlit tabanlı kullanıcı arayüzü, MongoDB destekli veri yönetimi ve Ollama entegrasyonu ile güçlendirilen sistem, yazılım test döngüsünde kritik öneme sahip senaryo oluşturma aşamasını daha hızlı, güvenilir ve ölçeklenebilir biçimde gerçekleştirmektedir. Çalışmanın geçerliliği, biri Python diğeri C++ tabanlı iki farklı proje üzerinde yapılan uygulamalarla doğrulanmıştır.**

Anahtar Kelimeler — Yazılım Test Otomasyonu, Büyük Dil Modelleri, Test Senaryosu Üretimi, Yapay Zeka Destekli Test

***Abstract*—Test scenario generation, one of the most critical phases in the Software Testing Life Cycle (STLC), often reduces overall efficiency due to its high time and resource requirements. As an innovative solution to this challenge, the Smart Test Scenario Tool (STS) has been developed. STS utilizes large language models to analyze documents in various formats, including xlsx, py, cpp, txt, and docx, thereby enabling the automated generation of contextually accurate test scenarios. This approach minimizes the time loss and human error risks inherent in traditional manual testing, while transforming the process into a more systematic and context-aware framework. The system, powered by a Streamlit-based interface, MongoDB-driven data management, and Ollama integration, ensures that the test scenario generation phase—crucial within the software testing lifecycle—is conducted with enhanced speed, reliability, and scalability. The effectiveness of the proposed system has been validated through two case studies implemented in Python- and C++-based projects, respectively.**

Keywords — Software Test Automation, Large Language Models, Test Scenario Generation, AI-Driven Testing

I. GİRİŞ

Test süreçlerinin otomatikleştirilmesi, hem test kapsamını genişletmek hem de manuel testlerde sıkça karşılaşılan zaman kaybı ile insan hatalarını en aza indirmek açısından önemli bir araştırma alanı hâline gelmiştir. Bu doğrultuda, büyük dil modellerine (LLM) dayalı test otomasyonu yaklaşımları son yıllarda dikkate değer ilerlemeler kaydetmiştir. Geleneksel yöntemlerin aksine, bu modeller daha derin bağlam farkındalığına sahip ve senaryo çeşitliliği açısından zengin test çıktıları üretebilmektedir. Örneğin, TESTPILOT sistemi, GPT-3.5-turbo ve Code-Cushman-002 modellerini kullanarak JavaScript programları için otomatik testler üretmekte ve test verimliliğini artırmayı amaçlamaktadır [3]. Benzer şekilde, Qodo aracı kaynak kod analizine dayalı olarak dağıtım öncesi hataları tespit etmekte; kapsamlı test senaryoları oluşturarak manuel test yükünü azaltmakta ve insan hatalarını önlemektedir [4]. LIBRO çerçevesi ise hata raporlarını analiz ederek test senaryoları türetmekte; Defects4J veri seti üzerinde yapılan deneylerde geliştiricilerin

hata teşhis yükünü kayda değer biçimde düşürmektedir [5]. Büyük dil modellerinin test senaryosu üretimindeki etkinliği, mobil uygulama testleri, platformlar arası uyumluluk ve kullanıcı arayüzü analizi gibi alanlarda da araştırılmıştır. Bu çalışmalarda, farklı platformlardan elde edilen kullanıcı arayüzü verilerinin işlenmesiyle test doğruluğunun anlamlı ölçüde arttığı gösterilmiştir [6]. CodaMosa algoritması da arama tabanlı yazılım testi (SBST) yöntemleriyle büyük dil modellerini birleştirerek test kapsamını genişletmeyi hedeflemekte; Codex destekli bu yaklaşımın klasik SBST yöntemlerine kıyasla daha yüksek kapsama oranlarına ulaştığı tespit edilmiştir [7]. Ayrıca, LLM tabanlı test otomasyonunun sürekli entegrasyon süreçleriyle uyumlu hale getirilmesi de araştırma konusu olmuş; "Test Durumu Seçimi ve Önceliklendirme (TSP)" tekniklerinin özellikle regresyon testlerinde sağladığı performans artışları raporlanmıştır [8]. ChatGPT'nin çevrimiçi test oluşturma ve değerlendirme aşamalarında kullanımı üzerine yapılan çalışmalar ise, bu modellerin eğitim teknolojilerinde test otomasyonu için yeni olanaklar sunduğunu, ancak belirli adımlarda insan denetimine hâlen ihtiyaç duyulduğunu göstermiştir [9]. Yapay zeka destekli test yaklaşımları içerisinde, özellikle AI sistemlerinin güvenilirliğini ölçmeye ve iyileştirmeye yönelik araştırmalar da önem kazanmıştır. CleanAI isimli kütüphane, derin sinir ağlarını nöron kapsama gibi metrikler üzerinden değerlendirerek beyaz kutu testlerini desteklemekte ve bu alana önemli katkılar sunmaktadır [10]. Benzer şekilde, GPT-3 destekli GUI test otomasyonu yöntemleri de doğal dil işleme tekniklerinden yararlanarak geliştirilmiş ve deneysel ortamlarda başarıyla doğrulanmıştır [11].

Akıllı Test Senaryosu Aracı (STS), test otonomisi yaklaşımını destekleyen bir sistem olarak yüksek performanslı donanım altyapısı ve güncel büyük dil modeli teknolojileriyle geliştirilmiştir. Sistemde Llama 3.2-3B modeli kullanılmakta olup, Intel i9-14900 işlemci, 65 536 MB RAM ve NVIDIA RTX 4000 Ada Generation grafik kartı gibi güçlü donanım bileşenlerinden yararlanılmaktadır. Ayrıca, Streamlit tabanlı kullanıcı arayüzü, MongoDB veri tabanı ve Ollama entegrasyonu ile modellerin yerel ortamda güvenli, verimli ve ölçeklenebilir biçimde çalışması sağlanmıştır. Bu yapı sayesinde STS, bağlamsal olarak tutarlı ve çeşitlilik açısından zengin test senaryoları üreterek geleneksel yöntemlerde görülen zaman kaybı ve hata oranlarını büyük ölçüde azaltmaktadır.

Çalışmanın devam eden bölümlerinde, Akıllı Test Senaryosu Aracı'nın (STS) mimarisi, kullanılan teknolojiler ve uygulama adımları ayrıntılı biçimde açıklanmakta; ardından test senaryosu oluşturma süreci ve elde edilen sonuçlar sunulmaktadır. Son bölümde ise, çalışmadan elde edilen çıkarımlar değerlendirilmekte ve gelecekteki araştırma yönelimleri tartışılmaktadır.

II. AKILLI TEST SENARYOSU ARACI (STS)

Yazılım test süreçleri, yazılım kalitesi ve güvenilirliği için kritik olup, manuel testlerin zaman, maliyet ve hata riskleri nedeniyle modern, yapay zeka destekli otomasyon çözümlerine ihtiyaç duymaktadır. Bu bağlamda, Akıllı Test Senaryosu Aracı (STS), büyük dil modellerini kullanarak dokümantasyonu analiz etmekte ve uygun test senaryoları oluşturarak bir test otonomisi sunmaktadır. ASRLAB, otonom sistemler, deneysel yazılım mühendisliği, performans ve güvenlik odaklı yapay zeka temelli test yaklaşımları gibi alanlarda teorik ve uygulamalı araştırmalara öncülük ederken, MATISSE Projesi, dijital ikizler ve model tabanlı mühendislik teknikleriyle endüstriyel sistemlerin doğrulama ve geçerleme süreçlerini optimize etmeyi amaçlayan Avrupa çapında bir projedir [12,13]. Bu kapsamda, Akıllı Test Senaryosu Aracı (STS), kullanıcı dokümanlarını analiz ederek JSON formatında test komutları üretmekte ve bu çıktıları MongoDB üzerinde güvenle saklarken, i9-14900 işlemci, 65536MB RAM ve NVIDIA RTX 4000 Ada Generation gibi yüksek performanslı donanım desteği ile sistem verimliliğini artırmaktadır (Şekil 1).

Şekil 1 ASRLab Fiziksel Geliştirme Ortamı

Akıllı Test Senaryosu Aracı (STS)'da kullanılan ISTQB standartlarına dayalı metodolojiler, oluşturulan test senaryolarının teorik ve pratik sağlamlığını garanti altına almaktadır [14].

A. Kullanılan Teknolojiler ve Araçlar

Akıllı Test Senaryosu Aracı (STS)'un etkin çalışmasını sağlamak üzere, bağlama uygun, detaylandırılmış ve standartlaştırılmış test senaryolarını otomatik olarak üretebilmek için Llama 3.2:3B modeli kullanılmakta; modellerin yerelde çalıştırılması, komut yönetimi ve yapısal çıktı elde edilmesi ise açık kaynaklı bir yazılım olan Ollama aracılığıyla gerçekleştirilmektedir [15]. Kullanıcıların doküman yükleme, test tiplerini belirleme ve senaryo özelleştirme gibi işlemleri kolayca yapabilmesi için etkileşimli bir Streamlit arayüzü sunulmakta [16], model çıktıları ve kullanıcı etkileşimleri ise JSON tabanlı esnek veri yapısına sahip MongoDB'de depolanarak oturumlar, dokümanlar ve senaryolar dinamik bir şekilde yönetilmektedir [17].

B. Test Senaryosu

Fonksiyonel ve fonksiyonel olmayan testler, yazılımın güvenilirlik, doğruluk ve performansını değerlendirmede temel araçlardır (Tablo 1). Akıllı Test Senaryosu Aracı (STS), bu test türlerini tek bir platformda birleştirerek büyük dil modelleriyle bağlamsal ve hızlı senaryo üretimi sayesinde otomasyon, zaman tasarrufu ve hata oranında azalma sağlamaktadır.

Tablo 1 Test Senaryosu Kategorileri ve Açıklamaları

Kategori	Açıklama
Fonksiyonel - Entegrasyon Testi	Bağlantılı modüller arasındaki veri akışını doğrular.
Fonksiyonel - Giriş Verisi Çeşitliliği Testi	Farklı format ve özellikteki giriş verilerinin sistem üzerindeki etkisini test eder.
Fonksiyonel - Fonksiyonel Test	Yazılımın belirlenen gereksinimlere uygun olarak çalışıp çalışmadığını değerlendirir.
Fonksiyonel - Kenar Durum ve Sınır Testi	Sınır ve uç durumlar karşısında sistemin dayanıklılığını test eder.
Fonksiyonel - Kullanıcı Arayüzü (GUI) Testi	Kullanıcı arayüzünün doğru çalıştığını ve kullanıcı deneyimini değerlendirir.
Fonksiyonel Olmayan - Performans ve Yük Testi	Kullanıcı yükü altında sistemin tepkisini değerlendirir.
Fonksiyonel Olmayan - Uyumluluk Testi	Yazılımın farklı donanım ve işletim sistemleriyle uyumluluğunu inceler.
Fonksiyonel Olmayan - Güvenlik Testi	Sistem güvenliğini değerlendirerek güvenlik açıklarını belirler.

Yazılım test süreçlerinin etkinliği, test senaryolarının doğruluk, bütünlük ve profesyonel standartlara uygunluk açısından değerlendirilmesine bağlıdır. Bu süreçte test puanlama öğeleri ve istemleri ile test talimat öğeleri ve istemleri, test senaryolarının kalitesini ve tutarlılığını belirleyen temel bileşenlerdir. Söz konusu bileşenler, test senaryolarının belirlenen standartlara uygunluğunu garanti etmek için sistematik ve nesnel bir değerlendirme süreci sunarak, test süreçlerinin daha yapılandırılmış ve güvenilir olmasını sağlar. Bu kapsamda test senaryolarının komutlarını özelleştirmeyi mümkün kılan Akıllı Test Senaryosu Aracı (STS) arayüzünde yapılan seçimlerin ana komuta nasıl etki ettiği Tablo 2'de gösterilmektedir.

Tablo 2. Test Senaryosu Bileşenleri

Bileşen	**İşlev**
Test Senaryosu Kalite Puanı	Test senaryosunun ilgililiğini, açıklığını ve derinliğini değerlendirir.
Çıktı Uyum Puanı	Oluşturulan senaryonun beklenen test yapısına ne kadar uyduğunu ölçer.
Tutarlılık Puanı	Senaryonun, hedefler, test adımları ve ölçütler arasında tutarlılığı koruyup korumadığını kontrol eder.
Detay ve Özellik Puanı	Senaryonun detay ve özgüllük seviyesini değerlendirir.
Profesyonel Standart Puanı	Senaryonun sektördeki en iyi uygulamalara ve profesyonel standartlara uygunluğunu sağlar.
Senaryo Bağlam Doğrulama	Senaryonun belirtilen test bağlamına ve amacına doğrudan uygun olup olmadığını doğrular.
İlgililik ve Tamlık Kontrolü	Gerekli tüm bileşenlerin (hedef, ön koşullar, kullanıcı yükü vb.) dahil olup olmadığını kontrol eder.
Tutarlılık ve Uyum Doğrulama	Senaryonun ilgili test kategorisiyle uyumunu sağlar ve tutarsızlıkları tespit eder.
Detay Doğrulama	Detay seviyesini gözden geçirir, eksik değerleri veya yeterince belirgin olmayan noktaları belirler.
Profesyonel ve Yapılandırılmış Biçimlendirme	Senaryonun yapısını ve netliğini değerlendirir, profesyonel biçimlendirme kurallarına uygunluğunu sağlar.

C. Test Senaryosu Oluşturma Süreci

Akıllı Test Senaryosu Aracı (STS)'un test senaryosu üretim sürecini adım adım açıklayan sözde kod yapısı sunulmaktadır. Akıllı Test Senaryosu Aracı (STS)'un test senaryosu üretim süreci, kullanıcı tarafından desteklenen formatlardan (örneğin: txt, docx, xlsx, py) bir dokümanın getUserDocument() fonksiyonu ile yüklenmesiyle başlar; ardından, languageModel.analyze() kullanılarak doküman büyük dil modeliyle analiz edilir ve doğal dil işleme teknikleriyle içerikteki anlamlı bileşenler ve test süreci için gerekli bilgiler belirlenir. determineTestType() fonksiyonu, bu analiz sonuçlarına dayanarak en uygun test türünü (fonksiyonel ya da fonksiyonel olmayan) seçip display() ile kullanıcıya sunar; kullanıcı ise getUserSelection() fonksiyonu aracılığıyla önerilen test türlerinden bir seçim yapar. Seçime bağlı olarak createCommand() fonksiyonu, test senaryosunun oluşturulması için gerekli komutu yapılandırır ve bu komut languageModel.execute() ile büyük dil modeline gönderilir. Gelen yanıt parseJSON() fonksiyonu kullanılarak JSON formatında ayrıştırılır, ardından saveToDatabase() ile veritabanına kaydedilir ve son olarak display() aracılığıyla kullanıcıya sunulur. Tüm adımlar Tablo 3'te sunulmaktadır.

Tablo 3. Akıllı Test Senaryosu Aracı (STS) İşlem Adımları

```
1. document = getUserDocument()
2. analysisResult = languageModel.analyze(document)
3. suggestedTest = determineTestType(analysisResult)
4. display("Önerilen test türü:", suggestedTest)
```

```
5.  userSelection = getUserSelection(suggestedTest)
6.  command = createCommand(userSelection)
7.  response = languageModel.execute(command)
8.  testScenario = parseJSON(response)
9.  saveToDatabase(testScenario)
10. display(testScenario)
```

Bu süreçte üretilen test senaryoları daha önceden belirlenmiş ve büyük dil modeline verilmiş JSON yapısına uygun şekilde senaryo kimliği, başlık, açıklama, hedef, kategori ve yorumlar gibi yapılandırılmış bileşenler içermektedir.

Akıllı Test Senaryosu Aracı (STS) kullanılarak TaskMaster projesine özel olarak 8 farklı test tipine göre test senaryoları üretilmiştir. TaskMaster, görev ve proje yönetimi için geliştirilmiş bir sistem olup, görevlerin oluşturulması, yönetilmesi ve takip edilmesine olanak tanımaktadır. Sistem, görev ekleme, silme, güncelleme ve tamamlama gibi temel işlevleri destekleyerek kullanıcıların projelerini organize etmelerini sağlar. Akıllı Test Senaryosu Aracı (STS) kullanılarak TaskMaster projesinde test işlemleri gerçekleştirilmiştir. Ayrıca, ProductDetection projesinde C++ kodu kullanılarak robotik sistemin uzaklık sensörlerini ölçen işlevin fonksiyonel test senaryoları üretilmiş ve elde edilen çıktıların kalitesi incelenmiştir.

D. Test Senaryosu Oluşturma Sonuçları

Akıllı Test Senaryosu Aracı (STS) kullanılarak, TaskMaster projesi için, Kenar Durum ve Sınır Testi, Fonksiyonel Test, Kullanıcı Arayüzü (GUI) Testi, Giriş Verisi Çeşitliliği Testi, Entegrasyon Testi, Uyumluluk Testi, Performans ve Yük Testi ve Güvenlik Testi için toplam 32 adet test senaryoları üretilmiştir (Tablo 4).

Tablo 4. Üretilen Test Senaryoları Tipleri ve Sayıları

Test Tipi	**Üretilen Test Senaryo Sayısı – Task Master**	**Üretilen Test Senaryo Sayısı – ProductionDetection**
Performans ve Yük Testi	2	-
Entegrasyon Testi	1	1
Giriş Verisi Çeşitliliği Testi	5	1
Fonksiyonel Test	1	1
Kenar Durum ve Sınır Testi	5	6
Uyumluluk Testi	1	-
Kullanıcı Arayüzü (GUI) Testi	6	3
Güvenlik Testi	11	-

Elde edilen test senaryoları örnek test senaryolarının bir arada gösterilmesi amacıyla TaskMaster'ında bulunduğu GitHub sayfasında bulunmaktadır [18]. Akıllı Test Senaryosu Aracı (STS) ile bu senaryolar, sistemin sınır koşulları, farklı giriş verileri, entegrasyon yapıları ve güvenlik açıkları karşısında nasıl çalıştığını değerlendirmeye yönelik oluşturulmuştur. Bu sayede, yazılımın Akıllı Test Senaryosu Aracı (STS) kullanımı ile üretilen test senaryolarının koşularak kalite ve güvenilirlik düzeyi artırılmasında daha etkin bir kullanıcı deneyimi sunulması sağlanmıştır. ProductDetection projesi ise kategori olarak fonksiyonel test senaryosu üretimine uygun olduğundan bu kapsamda 5 farklı test tipinde toplam 12 farklı test senaryosu oluşturulmuştur. TaskMaster ve ProductDetection projeleri için üretilen fonksiyonel test senaryolarının farkı tabloda gösterilmektedir (Tablo 5).

Tablo 5. TaskMaster ve ProductDetection İçin Oluşturulan Fonksiyonel Test Senaryosu Örnekleri

TaskMaster Fonksiyonel Test Senaryosu	{"ScenarioID": "fonksiyonel_test_senaryo_durumlari_function_testing_ornek_Test_Scenario_1", "Title": "Verify Task Creation and Editing Functionality", "Description": "This test scenario evaluates the system's ability to create, edit, and delete tasks. The goal is to ensure that users can successfully add new tasks with relevant details and update existing tasks with accurate information.", "Objective": "Validate task creation and editing functionality.", "Category": "Functional Testing", "Comments": ""}
ProductDetection Fonksiyonel Test Senaryosu	{"ScenarioID": "Process_A_ModelX_Test_Scenario_1", "Title": "Product Detection Functional Test Scenario", "Description": "This test scenario evaluates the ProductDetection class against specific functional requirements related to product detection based on sensor inputs and gripper state.", "Objective": "Validate the core functionalities of the ProductDetection class.", "Category": "Functional Testing", "Comments": ""}

Tabloda görüldüğü gibi STS, ona verilen girdilerin içeriklerine uygun şekilde test senaryolarını oluşturarak gereksiz ve gerçeklenemz test senaryolarının ilgili projeler için üretilmemesini sağlamaktadır.

III. SONUÇ VE GELECEK ÇALIŞMALAR

Akıllı Test Senaryosu Aracı (STS), test senaryolarının otomatikleştirilmesi sayesinde yazılım test mühendislerinin manuel görev yükünü önemli ölçüde azaltmış ve test süreçlerinin genel verimliliğini artırmıştır. Örneğin, geleneksel yöntemlerle 1000 test senaryosunun hazırlanması, günde 8 saat çalışan bir mühendis için, saatte ortalama 5 senaryo üretimi üzerinden yaklaşık 200 iş saati yani yaklaşık bir aylık çalışma süresi gerektirir. Oysa STS kullanıldığında bu süreç yalnızca birkaç dakika içerisinde tamamlanabilmektedir. Böylece, bir aylık manuel emek dakikalar içinde sonuçlanmakta, mühendisler de zamanlarını daha kritik analiz ve doğrulama görevlerine ayırabilmektedir. Gelecekte, test durumlarının otomatikleştirilmesinin kapsamı daha da genişletilerek test süreçlerinin hız ve etkinliğinde ek kazanımlar sağlanması hedeflenmektedir. Bu gelişmeler, yazılım testlerinin hem doğruluk hem de kalite açısından daha yüksek standartlara ulaşmasına katkı sağlayacaktır.

BİLGİLENDİRME

Proje, KDT Ortak Girişimi (101140216) (KDT JU) ve üyeleri tarafından desteklenmektedir. Bu destek, Vinnova (İsveç), Österreichische Forschungsförderungsgesellschaft mbH – FFG (Avusturya), Business Finland (Finlandiya), Üniversiteler ve Araştırma Bakanlığı (İtalya), FCT (Portekiz) ve TÜBİTAK (124N448) (Türkiye) tarafından sağlanan ek finansmanı da içermektedir.

KAYNAKLAR

[1] SmartBear Zephyr, SmartBear, [Online]. https://smartbear.com/test-management/zephyr/, 2025.

[2] Zaptest, Zaptest, [Online]. https://www.zaptest.com, 2025.

[3] M. Schafer, S. Nadi, A. Eghbali, and F. Tip, "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation," IEEE Transactions on Software Engineering, vol. 50, no. 1, pp. 85–105, Jan. 2024, doi: 10.1109/TSE.2023.3334955.

[4] Qodo AI, Qodo AI, [Online]. https://www.qodo.ai/, 2025.

[5] S. Kang, J. Yoon, and S. Yoo, "Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction," in Proceedings - International Conference on Software Engineering, IEEE Computer Society, 2023, pp. 2312–2323. doi: 10.1109/ICSE48619.2023.00194.

[6] S. Yu, C. Fang, Y. Ling, C. Wu, and Z. Chen, "LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities," in IEEE International Conference on Software Quality, Reliability and Security, QRS, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 206–217. doi: 10.1109/QRS60937.2023.00029.

[7] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "CodaMosa: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models," in Proceedings - International Conference on Software Engineering, IEEE Computer Society, 2023, pp. 919–931. doi: 10.1109/ICSE48619.2023.00085.

[8] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," Empir Softw Eng, vol. 27, no. 2, Mar. 2022, doi: 10.1007/s10664-021-10066-6.

[9] V. M. Ionescu and M. C. Enescu, "Using ChatGPT for Generating and Evaluating Online Tests," in 15th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2023 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/ECAI58194.2023.10193995.

[10] A. H. Ayubi, F. Taskin, O. Caglar, S. Asik, C. Baglum, and U. Yayan, "Evaluation of Deep Neural Network Quality by CleanAI Coverage Metrics Library," in 17th International Conference on INnovations in Intelligent SysTems and Applications, INISTA 2023 - Proceedings, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/INISTA59065.2023.10310324.

[11] Institute of Electrical and Electronics Engineers, 2023 IEEE 16th International Conference on Software Testing, Verification and Validation workshops 16-20 April 2023, Dublin, Ireland : proceedings.

[12] ASRLab OGU, ASRLab OGU, [Online]. https://srlab.ogu.edu.tr/, 2025.

[13] Matisse KDT, Matisse KDT, [Online]. https://matisse-kdt.eu/, 2025.

[14] ISTQB, ISTQB, [Online]. https://www.istqb.org/, 2025.

[15] Ollama, Ollama, [Online]. https://ollama.com, 2025.

[16] Streamlit, Streamlit, [Online]. https://streamlit.io, 2025.

[17] MongoDB, MongoDB, [Online]. https://www.mongodb.com, 2025.

[18] C. Bağlum, Smart Test Generation, [Online]. https://github.com/cembglm/Smart-Test-Generation, 2025.