T.C
OSTİM TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YAZILIM MÜHENDİSLİĞİ ANA BİLİM DALI
YAZILIM MÜHENDİSLİĞİ
YÜKSEK LİSANS PROGRAMI

ÜRETKEN YAPAY ZEKA DESTEKLİ YAZILIM PERFORMANS TESTLERİNİN HAZIRLANMASI VE ANALİZLERİNİN YAPILMASI

YÜKSEK LİSANS TEZİ

HAZIRLAYAN
BURAK TUZLUTAŞ

TEZ DANIŞMANI
Dr. Öğr. Üyesi MURAT ŞİMŞEK

ANKARA-2025

TEZ KABUL VE ONAY

Burak TUZLUTAŞ tarafından hazırlanan "Üretken Yapay Zeka Destekli Yazılım Performans Testlerinin Hazırlanması ve Analizlerinin Yapılması" başlıklı bu çalışma, 08/07/2025 tarihinde yapılan savunma sınavı sonucunda başarılı bulunarak jürimiz tarafından Yüksek Lisans olarak kabul edilmiştir.

Kabul Tarihi: 08/07/2025

Jüri Üyesi: **Dr. Öğr. Üyesi İclal ÇETİN TAŞ** _____
Başkent Üniversitesi

Jüri Üyesi: **Dr. Öğr. Üyesi Gülsüm KAYABAŞI KORU** _____
Ostim Teknik Üniversitesi

Tez Danışmanı: **Dr. Öğr. Üyesi Murat ŞİMŞEK** _____
Ostim Teknik Üniversitesi

ONAY

Jüri tarafından kabul edilen bu çalışmanın Yüksek Lisans Tezi olması için gerekli şartları yerine getirdiğini onaylıyorum.

23/07/2025

_____
Prof.Dr. Halil Rıdvan ÖZ
Enstitü Müdürü

BİLDİRİM

Enstitü tarafından onaylanan Yüksek Lisans tezimin tamamını veya herhangi bir kısmını basılı veya dijital biçimde arşivleme ve aşağıda belirtilen koşullar dahilinde erişime açma iznini Ostim Teknik Üniversitesine verdiğimi bildiririm. Bu izinle, Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak ve gelecekteki çalışmalar (makale, kitap, lisans, patent vb.) için tezimin tamamının veya bir bölümünün kullanım hakları yalnızca bana ait olacaktır.

Tezimin bütünüyle kendi çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Telif hakkı bulunan ve sahiplerinden yazılı izinle kullanılması zorunlu olan kaynakları, yazılı izin alarak kullandığımı ve istenildiğinde izinlerin suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayımlanan "Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge" kapsamında, tezim, aşağıda belirtilen koşullar haricince, YÖK Ulusal Tez Merkezi ve Ostim Teknik Üniversitesi Açık Erişim Sisteminde erişime açılır.

☐ Enstitü / Fakülte Yönetim Kurulu kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren 2 yıl ertelenmiştir.[1]

☐ Enstitü / Fakülte Yönetim Kurulunun gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren … ay ertelenmiştir.[2]

☐ Tezimle ilgili gizlilik kararı verilmiştir.[3][4]

Tarih
İmza

---

[1] MADDE 6(1) Lisansüstü tezle ilgili patent başvurusu yapılması veya patent alma sürecinin devam etmesi durumunda, tez danışmanının önerisi ve enstitü anabilim dalının uygun görüşü üzerine enstitü veya fakülte yönetim kurulu iki yıl süre ile tezin erişime açılmasının ertelenmesine karar verebilir.

[2] MADDE 6(2) Yeni teknik, materyal ve metotların kullanıldığı, henüz makaleye dönüşmemiş veya patent gibiyöntemlerle korunmamış ve internetten paylaşılması durumunda 3. şahıslara veya kurumlara haksız kazanç imkanı oluşturabilecek bilgi ve bulguları içeren tezler hakkında tez danışmanının önerisi ve enstitü anabilim dalının uygun görüşü üzerine enstitü veya fakülte yönetim kurulunun gerekçeli kararı ile altı ayı aşmamak üzere tezin erişime açılması engellenebilir.

[3] MADDE 7(1) Ulusal çıkarları veya güvenliği ilgilendiren, emniyet, istihbarat, savunma ve güvenlik, sağlık vb. konulara ilişkin lisansüstü tezlerle ilgili gizlilik kararı, tezin yapıldığı kurum tarafından verilir. Kurum ve kuruluşlarla yapılan iş birliği protokolü çerçevesinde hazırlanan lisansüstü tezlere ilişkin gizlilik kararı ise, ilgili kurum ve kuruluşun önerisi ile enstitü veya fakültenin uygun görüşü üzerine üniversite yönetim kurulu tarafından verilir. Gizlilik kararı verilen tezler Yükseköğretim Kuruluna bildirilir.

[4] MADDE 7(2) Gizlilik kararı verilen tezler gizlilik süresince enstitü veya fakülte tarafından gizlilik kuralları çerçevesinde muhafaza edilir, gizlilik kararının kaldırılması halinde Tez Otomasyon Sistemine yüklenir.

OSTİM TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ YÜKSEK LİSANS TEZİ ORİJİNALLİK RAPORU

Tezin Başlığı: Üretken Yapay Zeka Destekli Yazılım Performans Testlerinin Hazırlanması ve Analizlerinin Yapılması
Öğrencinin Adı, Soyadı: Burak TUZLUTAŞ
Tez Danışmanın Unvanı/Adı, Soyadı: Dr. Öğr. Üyesi Murat ŞİMŞEK
Anabilim Dalı: Yazılım Mühendisliği
Programı: Yüksek Lisans
Tarih: 23 / 07 / 2025

Yukarıda başlığı belirtilen Yüksek Lisans tez çalışmama ait Giriş, Ana Bölümler ve Sonuç kısmından oluşan ve toplam 89 sayfadan ibaret olan kısmı, 23/07/2025 tarihinde tez danışmanım tarafından Turnitin intihal tespit programında incelenmiştir. Orijinallik raporunda aşağıda ifade edilen filtrelemeler uygulanmıştır.

Orijinallik raporuna göre, tezimin benzerlik oranı % 3'dür.
Uygulanan filtrelemeler:

1. Kaynakça (hariç)

2. Alıntılar (hariç)

3. Beş (5) kelimeden daha az örtüşme içeren metin kısımları (hariç)

Tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrencinin İmzası:

ONAY
Tarih: 23 / 07 / 2025
Dr. Öğr. Üyesi Murat ŞİMŞEK

ETİK BEYAN

Bu çalışmanın özgün bir çalışma olduğunu, çalışmanın hazırlık, veri toplama, analiz, bilgilerin sunumu ve diğer tüm aşamalarında bilimsel etik ve kurallara uygun davrandığımı, çalışmada bulunan tüm belge bilgileri akademik etik ve kurallar çerçevesinde elde ettiğimi,görsel, işitsel ve yazılı bütün bilgileri ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu, kullanmış olduğum verilerde herhangi bir tahrifat yapmadığımı, yararlandığım kaynaklara bilimsel normlara uygun olarak atıfta bulunduğumu, tezimin kaynak gösterdiğim durumlar dışında tarafımdan kaleme alındığını ve özgün olduğunu, Tez danışmanım Dr. Öğr. Üyesi Murat ŞİMŞEK danışmanlığında ve tarafımdan üretildiğini ve OSTİM Teknik Üniversitesi Tez Yazım Kılavuzuna uygun olarak yazıldığını beyan ederim.

BURAK TUZLUTAŞ

23 / 07 / 2025

TEŞEKKÜR

Akademik yolculuklar yalnızca bilgiyle değil; inanç, destek ve sabırla da şekillenir. Bu yolculukta yanımda olan, beni yolda tutan ve vazgeçmeme izin vermeyen tüm kıymetli insanlara içten teşekkürlerimi sunuyorum.

Bu yolculukta, her adımımda fikirlerime değer veren, düşünme biçimimi geliştiren, beni sadece yönlendiren değil aynı zamanda düşünmeye teşvik eden danışman hocam **Dr. Öğr. Üyesi Murat ŞİMŞEK**'e en içten teşekkürlerimi sunuyorum. Her görüşmemizde zamanını ayırarak beni sabırla dinlemesi, sorularıma açık yüreklilikle yaklaşması ve bana duyduğu güvenle bu süreci daha güçlü ve kararlı bir şekilde sürdürmemi sağlaması, bu tezin ilerlemesinde çok değerli bir rol oynamıştır. Kendisinin akademik duruşu, anlayışı ve yol göstericiliği; yalnızca bu çalışmanın değil, benim de gelişimimin önemli bir parçası olmuştur. Tez danışmanım olarak katkısı ve emeği benim için daima özel bir anlam taşıyacaktır.

Bu süreçte bana destek olan ekip arkadaşlarıma da teşekkürlerimi sunmak isterim. Özellikle Ethem Utku Aktaş, Erhan Tayar, Burak Yeşiltaş ve Serhat Gökalp Özdemir araştırmanın farklı aşamalarında sağladıkları katkılar ve paylaştıkları bilgilerle bu çalışmanın gelişimine katkıda bulundular. Kendileriyle birlikte çalışmak büyük bir ayrıcalıktı.

Hayatımın en kıymetlisi, sabrın ve sevginin adı olan **eşim Şeyda TUZLUTAŞ ve ailem**… Bu süreçte gösterdiğiniz anlayış, taşıdığınız yük ve verdiğiniz güç olmasaydı, bugün bu sayfalara ulaşmak mümkün olmazdı. Varlığınız, her şeyin ötesinde bir şanstı. Minnettarım.

Ayrıca bu iki yıllık süreç boyunca bilgi birikimlerini paylaşan, düşünmeye ve üretmeye teşvik eden tüm hocalarıma teşekkürlerimi sunarım. Her katkınız, bu çalışmanın bir parçasıdır.

Ve elbette… Yolun başında kendine söz verip, yorgunluklara, zorluklara rağmen o söze sadık kalan kendime de teşekkür ediyorum. Bu çalışma yalnızca bir akademik emek değil, aynı zamanda kişisel bir direncin ve inancın ürünüydü.

BURAK TUZLUTAŞ

Tarih: 23/07/2025

ÖZ

Yazar Adı ve Soyadı	: BURAK TUZLUTAŞ
Üniversite	: OSTİM Teknik Üniversitesi
Enstitü	: Fen Bilimleri Enstitüsü
Program Adı	: Yazılım Mühendisliği
Tezin Türü	: Yüksek Lisans Tezi
Sayfa Sayısı	: 89
Tarihi	: 2025

ÜRETKEN YAPAY ZEKA DESTEKLİ YAZILIM PERFORMANS TESTİ OTOMASYONU

Bu tez çalışması, yazılım performans testi süreçlerinin otomasyonu ve verimliliğinin artırılması amacıyla üretken yapay zekâ (Generative AI) teknolojilerinin uygulanabilirliğini araştırmaktadır. Geleneksel performans testleri, test senaryosu oluşturma, test verisi üretimi, yük testi koşumu ve sonuç analizi gibi aşamalarda ciddi insan müdahalesi, teknik uzmanlık ve zaman gerektirmektedir. Bu bağlamda tezde; API'lerin otomatik keşfi, test senaryolarının JMeter gibi araçlara uygun biçimde üretimi, test profillerinin belirlenmesi ve yapay zeka destekli analiz süreçlerinin bir bütün olarak optimize edilmesi hedeflenmiştir. Araştırma kapsamında, hem açık kaynak yerel Büyük Dil Modelleri çözümleri (ör. LLaMA.cpp, Ollama) hem de bulut tabanlı sistemler (ör. OpenAI GPT-4) kullanılarak; Postman collection'larından otomatik olarak JMeter test planı olan .jmx senaryoları oluşturulmuş, yük testleri koşulmuş, test sonuçları sistem kaynak metrikleriyle entegre edilerek analiz edilmiştir. Deneysel süreçlerde, AI tabanlı model çıktıları hem test doğruluğunu hem de senaryo çeşitliliğini artırmış, aynı zamanda zamandan ve insan kaynağından önemli ölçüde tasarruf sağlandığı görülmüştür. Sonuçlar, üretken yapay zekâ destekli çözümlerin performans testi alanında yalnızca yardımcı bir unsur değil, sürecin merkezi bir otomasyon katmanı olarak konumlanabileceğini ortaya koymaktadır. Bu tez, özellikle yapay zekâ temelli performans testi otomasyonu alanında gerek akademik gerekse endüstriyel uygulamalara yönelik özgün bir katkı sunmaktadır.

Anahtar Sözcükler: Yazılım Performans Testi, Test Otomasyonu, Üretken Yapay Zeka, JMeter, API Testi, LLM, GPT-4, LLaMA

ABSTRACT

Thesis	: BURAK TUZLUTAŞ
University	: OSTİM Technical University
Institute	: Graduate School of Natural and Applied Sciences
Program's Name	: Software Engineer
Thesis Type:	: Master
Pages	: 89
Year	: 2025

GENERATIVE AI-ASSISTED SOFTWARE PERFORMANCE TEST AUTOMATION

This thesis investigates the applicability of generative AI technologies to automate and improve the efficiency of software performance testing processes. Traditional performance testing requires significant human intervention, technical expertise and time for test case generation, test data generation, load test execution and result analysis. In this context, the thesis aims to optimize the automatic discovery of APIs, generation of test cases in accordance with tools such as JMeter, determination of test profiles and AI-supported analysis processes as a whole. Within the scope of the research, using both open source native Large Language Model solutions (e.g. LLaMA.cpp, Ollama) and cloud-based systems (e.g. OpenAI GPT-4), .jmx scenarios with JMeter test plans were automatically generated from Postman collections, load tests were run, and test results were analyzed by integrating with system resource metrics. In the experimental processes, AI-based model outputs increased both test accuracy and scenario diversity, while at the same time, significant savings in time and human resources were achieved. The results show that generative AI-powered solutions can be positioned as a central automation layer of the process, not just an auxiliary element in the field of performance testing. This thesis makes an original contribution to both academic and industrial applications, especially in the field of AI-based performance testing automation.

Keywords: Software Performance Testing, Test Automation, Generative AI, JMeter, API Testing, LLM, GPT-4, LLaMA

İÇİNDEKİLER

TABLOLAR DİZİNİ

ŞEKİLLER DİZİNİ

SİMGELER VE KISALTMALAR

CI/CD	Sürekli Entegrasyon ve Sürekli Dağıtım (Continuous Integration and Continuous Deployment)
GEN AI	Üretken Yapay Zekâ (Generative Artificial Intelligence)
YA	Yapay Zeka (Artificial Intelligence)
LLM	Büyük Dil Modeli (Large Language Model)
API	Uygulama Programlama Arayüzü (Application Programming Interface)
JMX	JMeter XML
LocalLLM	Yerel Büyük Dil Modeli (Local Large Language Model)
TDD	Test Odaklı Yazılım Geliştirme (Test Driven Development)
UAT	Kullanıcı Kabul Testi (User Acceptance Testing)
TPS	Saniye Başına İşlem Sayısı (Transactions Per Second)
RPS	Saniyedeki İstek Sayısı (Requests Per Second)
IDE	Entegre Geliştirme Ortamı (Integrated Development Environment)
XSS	Siteler Arası Komut Dosyası Çalıştırma (Cross-Site Scripting)
WCAG	Web İçeriği Erişilebilirlik Yönergeleri (Web Content Accessibility Guidelines)
NLP	Doğal Dil İşleme(Natural Language Processing)
SEO	Arama Motoru Optimizasyonu(Search Engine Optimization)

1. GİRİŞ

Bu tez çalışması, yazılım mühendisliği disiplininde performans testlerinin uçtan uca otomasyonu amacıyla Üretken Yapay Zekâ (Generative AI- GenAI) teknolojilerinin entegrasyonunu ele almaktadır. Geleneksel yazılım performans test süreçleri; yüksek teknik uzmanlık gereksinimi, kapsamlı senaryo yapılandırmaları, statik test verisi yönetimi ve karmaşık sonuç analizleri nedeniyle zaman ve kaynak açısından verimsizdir. Özellikle mikroservis tabanlı, yüksek trafikli ve ölçeklenebilir yazılım sistemlerinde performans testlerinin başarıyla yürütülmesi, test kapsamının doğruluğu, senaryo gerçekçiliği ve analiz güvenilirliği gibi temel kriterlerle doğrudan ilişkilidir. Bu bağlamda, yapay zekâ destekli otomasyon yapıları, test süreçlerinin hızlandırılması, hataların azaltılması ve test mühendisliği pratiklerinin erişilebilir hale getirilmesi açısından büyük potansiyel sunmaktadır.

Tez kapsamında önerilen sistem mimarisi, Büyük Dil Modelleri (Large Language Models – LLM), özellikle GPT-4, GPT-3.5 ve LLaMA gibi mimarilerin yönlendirmeli kullanımı üzerine kuruludur. Girdi olarak kullanılan doğal dil tabanlı prompt'lar sayesinde; Postman Collection ya da Swagger/OpenAPI şemalarından API uç noktaları otomatik olarak toplanmakta, buradan test edilecek servisler analiz edilmekte ve otomatik olarak JMeter .jmx dosyaları oluşturulmaktadır. Ayrıca, test verilerinin üretimi için Mockaroo, Faker.js, DataFaker gibi araçlar ya da doğrudan AI model çıktıları kullanılmakta; response chaining ve auth gibi bağımlı yapılar otomatik işlenerek pre/post processor blokları oluşturulmaktadır.

Bu sistem, yerel (offline) çalışma desteği sunan LocalLLM çözümleri (ör. LLaMA.cpp, Ollama) ile de uyumlu şekilde tasarlanmıştır. Böylece sistem, ağ bağlantısının olmadığı ya da veri gizliliği gereksiniminin bulunduğu kurumsal ortamlarda da güvenle kullanılabilir hâle getirilmiştir. Kullanıcı arayüzü olarak, Python tabanlı Gradio kütüphanesiyle geliştirilen etkileşimli bir web arayüzü sunulmuş; kullanıcıdan alınan kısa açıklamalarla test dosyaları oluşturulup, test yürütülmekte ve sonuçlar doğal dil ile analiz edilmektedir.

Sistemin yapısı iki temel GenAI kullanım senaryosuna dayanmaktadır: (1) Performans testi senaryolarının üretimi ve konfigürasyonu, (2) Test yürütüldükten sonra sonuçların .jtl dosyası, sistem metrikleri (CPU, RAM, I/O, latency vs.) ve Grafana/InfluxDB gibi araçlardan elde edilen izleme verileriyle birlikte analiz edilerek doğal dilde açıklamalar ve önerilerle raporlanmasıdır. Bu yapı yalnızca manuel çabayı azaltmakla kalmamış, aynı

zamanda test çıktılarının daha okunabilir, anlamlandırılabilir ve aksiyona dönüştürülebilir hale gelmesini sağlamıştır.

Deneysel bölümde, önerilen yaklaşım geleneksel test mühendisliği yöntemleriyle karşılaştırmalı olarak değerlendirilmiş; hazırlık süresi, senaryo bütünlüğü, testin doğruluk oranı, response analiz kapasitesi ve hata tespit oranları gibi metrikler göz önüne alınmıştır. Bulgular, üretken yapay zekâ destekli test sistemlerinin geleneksel yöntemlere kıyasla %50'ye varan zaman tasarrufu ve daha düşük hata oranı sunduğunu göstermektedir. Ayrıca AI tabanlı analiz sistemleri, özellikle test sonrası sistem davranışlarının kök neden analizinde yüksek doğrulukla tespitler yapabilmiştir.

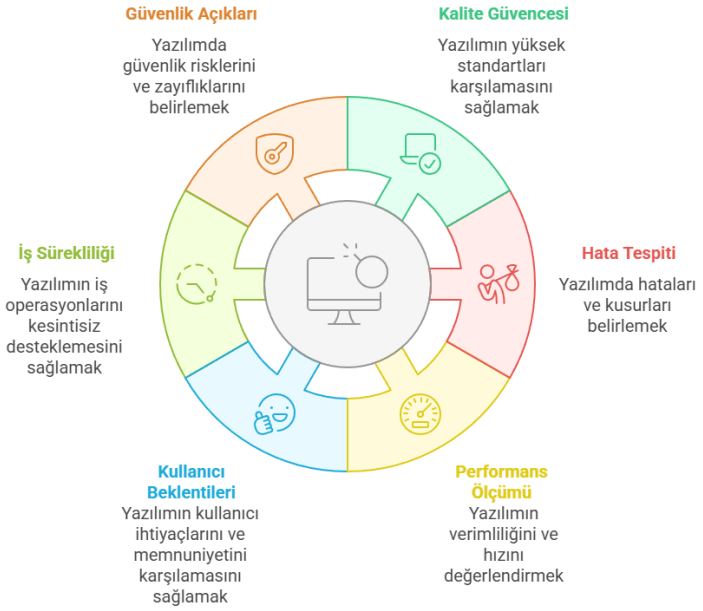
Bu tez, yalnızca yazılım test süreçlerini otomatikleştirme amacıyla değil, aynı zamanda test mühendisliğine yeni bir paradigma kazandırma hedefiyle yapılandırılmıştır. Özellikle Büyük Dil Modeli mimarilerinin test stratejisi üretimi, performans profili çıkarımı ve anomali analizi gibi alanlarda konumlanması, yazılım mühendisliğinin geleceğinde daha özerk ve akıllı sistemlere geçişi temsil etmektedir.

2. KAVRAMSAL ÇERÇEVE VE LİTERATÜR İNCELEMESİ

2.1 Kavramsal Çerçeve

2.1.1.Yazılım Testi ve Önemi

Yazılım testi, bir yazılım ürününün kalitesini güvence altına almak, hataları tespit etmek, gereksinimleri karşıladığını doğrulamak ve performansını ölçmek amacıyla gerçekleştirilen sistematik bir mühendislik sürecidir. Bu süreç, yalnızca hataların bulunmasıyla sınırlı kalmayıp, aynı zamanda sistemin kullanıcı beklentilerine uygunluğunu ve işlevsel bütünlüğünü de değerlendirmeyi kapsar. IEEE'ye göre yazılım testi; doğrulama, geçerleme ve değerlendirme süreçlerini kapsayan etkinliklerin tamamıdır.[1]. Şekil 2.1, yazılım testinin güvenlikten performansa çok yönlü etkisini göstermektedir.



Şekil 2.1. Yazılım Testinin Çok Yönlü Rolü

Modern yazılım geliştirme döngüsünde test, son aşamada devreye giren bir mekanizma değil, yazılımın yaşam döngüsüne entegre edilen sürekli bir kalite güvencesi yaklaşımına dönüşmüştür. Agile, DevOps ve CI/CD (Continuous Integration / Continuous Delivery) gibi modern yazılım geliştirme metodolojileri, yazılım testlerinin yalnızca manuel testlerle değil, otomatikleştirilmiş ve sürekli çalışan testlerle gerçekleştirilmesini gerekli kılmıştır [2]. Bu dönüşümle birlikte test süreçleri artık yazılım geliştirme sürecinin kalbi haline gelmiş, kod kalitesi kadar iş süreçlerinin sürekliliği, müşteri memnuniyeti ve sistemin sürdürülebilirliği de test kalitesine bağımlı hale gelmiştir.

Yazılım testinin temel amacı, yazılımın işlevsel ve işlevsel olmayan (non-functional) tüm gereksinimlerini karşılayıp karşılamadığını değerlendirmektir. Bu gereksinimler; sistemin doğru sonuç üretmesi, yeterli performansı sağlaması, güvenlik açıklarını taşımaması, farklı platformlarda sorunsuz çalışması ve kullanıcı arayüzünün sezgisel olması gibi geniş bir yelpazeyi kapsar. Ayrıca, test süreci sonucunda elde edilen bilgiler, yazılım geliştirme sürecinin sonraki aşamalarında yapılacak iyileştirmeler için kritik bir geri bildirim mekanizması olarak işlev görür.

Yazılım testi, aynı zamanda ürün güvenliğinin ve itibari değerinin korunmasında da önemli rol oynar. Özellikle sağlık, savunma, finans gibi yüksek hassasiyet gerektiren sektörlerde test süreçlerinin başarısı, doğrudan kullanıcıların yaşamını ve güvenliğini etkileyebilmektedir. Örneğin; 2015 yılında ABD'de Boeing 787 Dreamliner uçaklarında yazılım hatası nedeniyle yaşanan sistem kapanmaları, test süreçlerinin ne denli hayati olduğunun çarpıcı bir örneğidir [3].

Gelişen teknolojilerle birlikte yazılım testine dair yaklaşımlar da değişmektedir. Geleneksel test yöntemlerinin yerini, yapay zeka destekli otomasyon sistemleri, model tabanlı test araçları, üretken yapay zeka tabanlı test senaryosu oluşturma sistemleri almaya başlamıştır. Bu değişim hem maliyetleri azaltmakta hem de test kapsamını ve hızını artırarak, yazılım kalitesini daha yüksek seviyelere taşımaktadır [4].

Sonuç olarak, yazılım testi yalnızca teknik bir gereklilik değil; bir kalite güvencesi, kullanıcı memnuniyetini sağlayan bir strateji ve rekabet avantajı yaratan bir süreç olarak görülmelidir. Bu bağlamda yazılım testi, günümüz bilişim dünyasında sürdürülebilir ve güvenilir yazılım geliştirme hedeflerinin vazgeçilmez bir bileşeni haline gelmiştir.

Yazılım testi, günümüz yazılım mühendisliği süreçlerinde yalnızca bir kalite kontrol aracı değil; aynı zamanda riskleri azaltma, maliyetleri düşürme, kullanıcı memnuniyetini artırma ve yazılımın sürdürülebilirliğini sağlama noktasında stratejik bir gerekliliktir. Yazılım projeleri büyüdükçe ve daha karmaşık hâle geldikçe, test süreçlerinin yalnızca son aşamalara bırakılması kabul edilemez hâle gelmiştir. Bunun yerine test, yazılım yaşam döngüsünün her aşamasında yer almalı ve sistematik biçimde yürütülmelidir [5], [7].

Hataların Erken Tespiti: Yazılım geliştirme sürecinde hataların erken evrede tespit edilmesi, proje maliyetlerini önemli ölçüde azaltır. Boehm'in ortaya koyduğu "Hata Düzeltme Maliyeti Eğrisi", bir hatanın üretim aşamasında tespit edilmesinin, gereksinim aşamasına göre yaklaşık 100 kat daha maliyetli olduğunu göstermektedir [3]. Bu bağlamda testin yalnızca kodlama sonrası değil, analiz ve tasarım aşamalarında da uygulanması gereklidir. Özellikle white-box ve unit test yaklaşımları, hataların bileşen düzeyinde erken yakalanmasına olanak tanır [5].

Yazılım Kalitesinin Artırılması: Yazılım testi, yalnızca hataları bulmakla kalmaz; aynı zamanda doğruluk, kararlılık, güvenilirlik, taşınabilirlik ve kullanılabilirlik gibi temel kalite niteliklerini ölçme ve doğrulama imkânı sunar. Myers, yazılım testini, bir ürünün yalnızca doğru çalışıp çalışmadığını değil, aynı zamanda "beklenmeyen durumlarda nasıl davrandığını" da analiz etmenin aracı olarak tanımlar [5]. Test senaryolarının kapsamlı olması, yazılımın farklı kullanım durumlarına karşı dayanıklılığını artırır. Otomatik testler ise bu kaliteyi sürdürülebilir kılar [10].

Bakım ve Destek Maliyetlerinin Azaltılması: Bakım maliyetleri, genellikle yazılımın canlı ortama geçmesinden sonra karşılaşılan hatalar nedeniyle artar. Bu hatalar çoğunlukla testin yetersizliğinden kaynaklanır. IEEE 829 standardı, test planı, test vaka dokümanı, hata raporu ve sonuç analizlerinin sistematik biçimde belgelenmesini önererek bu riskleri minimize etmeyi amaçlar [8]. İyi tanımlanmış test süreçleri, yalnızca yazılımın güvenilirliğini değil, bakım süreçlerinin öngörülebilirliğini de artırır.

Kullanıcı Memnuniyetinin Artırılması: Yazılım sistemlerinin başarısı, yalnızca iç yapılarının sağlamlığına değil, kullanıcıyla olan etkileşimin kalitesine de bağlıdır. Kullanıcı arayüzünde ortaya çıkan hatalar, karmaşık hata raporlamaları ya da sistemin yavaş yanıt vermesi, kullanıcı deneyimini doğrudan olumsuz etkiler. Kaner ve arkadaşlarına göre, test yalnızca teknik bir faaliyet değil, aynı zamanda kullanıcı memnuniyetini artıran bir süreçtir [6]. Kullanılabilirlik ve performans testleri, bu noktada önemli rol oynar.

Güvenlik Açıklarının Azaltılması: Artan siber tehditler ve veri ihlalleri nedeniyle, güvenlik testleri yazılım test sürecinin ayrılmaz bir parçası hâline gelmiştir. Özellikle kimlik doğrulama, veri bütünlüğü ve erişim kontrolü gibi alanlarda yapılacak testler, güvenlik açıklarını minimize eder. ISO/IEC/IEEE 29119-1 standardı, güvenlik testlerinin sistemli biçimde gerçekleştirilmesini ve test sürecine entegre edilmesini önerir [9]. Bu sayede yazılım yalnızca işlevsel değil, güvenli de hâle gelir.

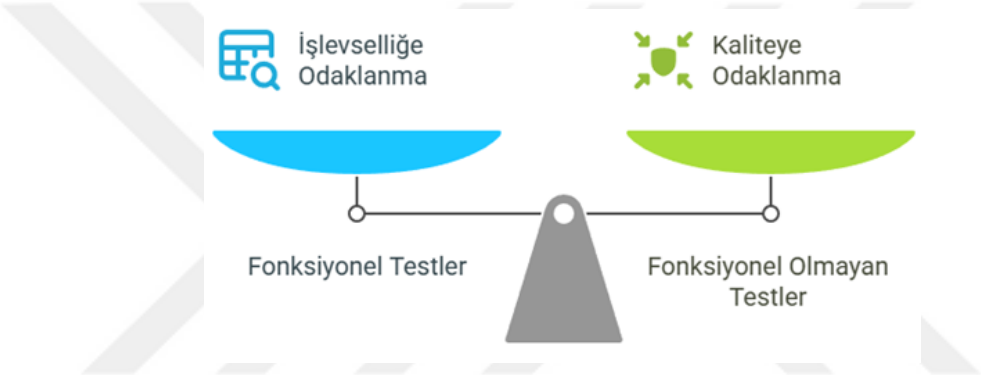
Yasal Uyumluluk ve Endüstri Standartlarına Uyum: Yazılım projeleri yalnızca teknik değil, aynı zamanda yasal yükümlülüklere de tabidir. Otomotiv (ISO 26262), sağlık (IEC 62304) ve havacılık (DO-178C) gibi yüksek riskli sektörlerde geliştirilen yazılımlar, uluslararası standartlara uygun test süreçlerine sahip olmak zorundadır. ISO/IEC/IEEE 29119-1 standardı bu kapsamda, test yönetimi ve süreci için çerçeve sağlar [5]. Yasal uyum, testin sistematik olarak planlanmasını ve izlenebilirliğini zorunlu kılar.

Sürekli Entegrasyon ve Teslimat Süreçlerinin Desteklenmesi: Agile, DevOps ve CI/CD gibi modern yazılım geliştirme metodolojileri, test süreçlerinin otomatik ve sürekli biçimde

yürütülmesini gerekli kılar. Her kod değişikliği sonrası gerçekleştirilen otomatik testler, entegrasyon sırasında sistemin bozulmadığını doğrular. Pressman ve Maxim, sürekli entegrasyonun başarıya ulaşmasının temel şartlarından birinin, kapsamlı test stratejileri olduğunu vurgular [10]. Bu yaklaşım sayesinde teslimat süresi kısalıri sistem güvenliği artar.

2.1.2. Yazılım Test Türleri

Yazılım test türleri genel olarak iki ana gruba ayrılır: **Fonksiyonel testler** ve **fonksiyonel olmayan testler**. Bu sınıflandırma, testin yazılımın "ne yaptığına" mı yoksa "nasıl yaptığına" mı odaklandığını belirler [5], [10], [14]. Şekil 2.2'de gösterildiği gibi fonksiyonel testler, sistemin gereksinimlere uygun işlevleri yerine getirip getirmediğini değerlendirirken; fonksiyonel olmayan testler, sistemin performansı, güvenliği, kullanılabilirliği gibi kalite niteliklerini ölçer [5], [7], [13].



Şekil 2.2. Yazılım Testlerinde İşlevsellik ve Kaliteyi Dengeleme

Fonksiyonel Testi: bir yazılım sisteminin işlevsel gereksinimlere uygunluğunu doğrulamak için yürütülen bir test türüdür. Bu test yaklaşımı, sistemin **"ne yaptığı"** sorusuna odaklanır ve çoğunlukla **siyah kutu testi** (black-box testing) olarak uygulanır. Yani test uzmanı, kodun iç işleyişine odaklanmadan, dış gözlemle (girdi-çıktı kontrolü) yazılımın davranışlarını sınar [5]. Test senaryoları, yazılımın işlevsel gereksinimleri temel alınarak hazırlanır ve genellikle kullanıcı arayüzü, API'ler, veri işleme mekanizmaları gibi fonksiyonel bileşenler üzerinde gerçekleştirilir [7].

Fonksiyonel test, yalnızca hataların tespit edilmesi için değil, aynı zamanda müşteri gereksinimlerinin sağlandığını kanıtlamak, kabul kriterlerini doğrulamak ve yazılım kalitesini güvence altına almak için de kullanılır [6]. Şekil 2.3, fonksiyonel testin birimden sisteme, entegrasyondan regresyona kadar tüm temel bileşenlerini göstermektedir.

Şekil 2.3. Fonksiyonel Testin Temel Bileşenleri

Birim (Unit) Testi: Birim testi, yazılımın en küçük yapı taşı olan fonksiyon veya metodun bağımsız şekilde test edilmesidir. Genellikle geliştiriciler tarafından kod yazımı sırasında yürütülür. Amaç, fonksiyonun beklenen çıktıyı üretip üretmediğini kontrol etmektir [5]. Bu test türü, test odaklı yazılım geliştirme (TDD) gibi yaklaşımlarda temel bir rol oynar. Otomasyon araçlarıyla (JUnit, NUnit gibi) yaygın olarak desteklenir. Hataların erken tespitiyle yazılımın modülerliğini ve bakım kolaylığını artırır [7].

Entegrasyon (Integration) Testi: Birimler arası veri akışı ve işbirliği test edilir. Amaç, farklı modüllerin birlikte çalışabilirliğini sağlamaktır [7], [6], [13]. Veri tutarlılığı sağlanır, sistemsel hatalar erken aşamada bulunur [6]. API testlerinde ve servis entegrasyonlarında kullanılır. Mock servisler ile entegrasyon senaryoları test edilir [13], [14]. Gerçek çalışma ortamına yakın test sağlar, sistem entegrasyon sorunlarını azaltır.

Sistem (System) Testi: Tüm yazılım sisteminin entegre olarak test edilmesini kapsar. Uçtan uca test senaryoları oluşturularak, gereksinimlerin eksiksiz yerine getirilip getirilmediği doğrulanır. Genellikle bağımsız test ekipleri tarafından gerçekleştirilir [10]. Sistem testi, hem fonksiyonel hem de bazı fonksiyonel olmayan gereksinimleri kapsayabilir ve yazılımın gerçek çalışma ortamına yakın koşullarda test edilmesini sağlar [6].

Kabul Testi: yazılımın müşteriye teslim edilmeden önce iş gereksinimlerini karşılayıp karşılamadığını değerlendirir. Genellikle müşteri temsilcisi veya son kullanıcı tarafından gerçekleştirilir. Bu testin sonucunda yazılımın "kabul" edilip edilmeyeceği belirlenir [9]. Kullanıcı Kabul Testi (UAT) süreci, projelerin yasal, operasyonel ve işlevsel olarak tamamlandığını onaylamak açısından kritiktir.

Regresyon Testi: yapılan kod değişikliklerinin sistemin mevcut işlevselliğini olumsuz etkileyip etkilemediğini anlamak için yapılır. Bu testler özellikle sürekli entegrasyon (CI) ortamlarında önemlidir [11]. Yaygın olarak otomatikleştirilir ve Selenium, TestNG, Cypress gibi araçlarla yürütülür. Testlerin yeniden çalıştırılabilir olması, zaman ve maliyet tasarrufu sağlar [7].

Duman Testi: sistemin temel işlevlerinin çalışıp çalışmadığını hızlıca doğrulamak için yapılır. "Yazılım yanıyor mu?" metaforuna dayanan bu test türü, genellikle yeni sürüm yüklemelerinden sonra ilk yapılan testtir [12]. Temel stabilite kontrolü sağlayarak daha detaylı testlere geçilip geçilmeyeceğini belirler.

Arayüz Testleri: bir yazılım sistemi içerisindeki bileşenlerin veya dış sistemlerin (örneğin servisler veya API'ler) birbiriyle olan iletişimini sınar. Veri bütünlüğü ve protokol uyumu gibi kriterler test edilir [13]. Bu testler, Postman veya RestAssured gibi araçlarla API düzeyinde yaygın olarak yürütülür.

Geriye Dönük Uyumluluk (Backward Compatibility) Testi: yeni yazılım sürümünün eski verilerle ya da sistemlerle uyumlu çalışıp çalışmadığını sınar. Özellikle büyük kurumsal sistemlerde, kullanıcı deneyiminin kesintisiz sürdürülebilmesi için bu testler kritik öneme sahiptir [14]. Kullanıcı verisi kaybını önler, sürüm geçişlerinde güvenli geçiş sağlar.

Fonksiyonel olmayan test türleri, yazılımın "ne" yaptığına değil, "nasıl" yaptığına odaklanır. Performans, güvenlik, kullanılabilirlik, taşınabilirlik, uyumluluk gibi sistemin nitel özelliklerini değerlendirir [5], [6], [10]. Bu testler, kullanıcı deneyimini doğrudan etkileyen unsurları ölçerek sistemin kalitesini artırmayı amaçlar. Yalnızca sistemin işlevselliği değil; sistemin ölçeklenebilirliği, kaynak verimliliği, güvenli kullanım olanakları, geniş kitlelere hitap edebilmesi ve engelli bireylerin erişebilmesi gibi çok sayıda hayati özellik bu testlerle ölçümlenebilir. Bu sayede yazılım, sadece çalışan değil aynı zamanda sürdürülebilir ve erişilebilir hale gelir.

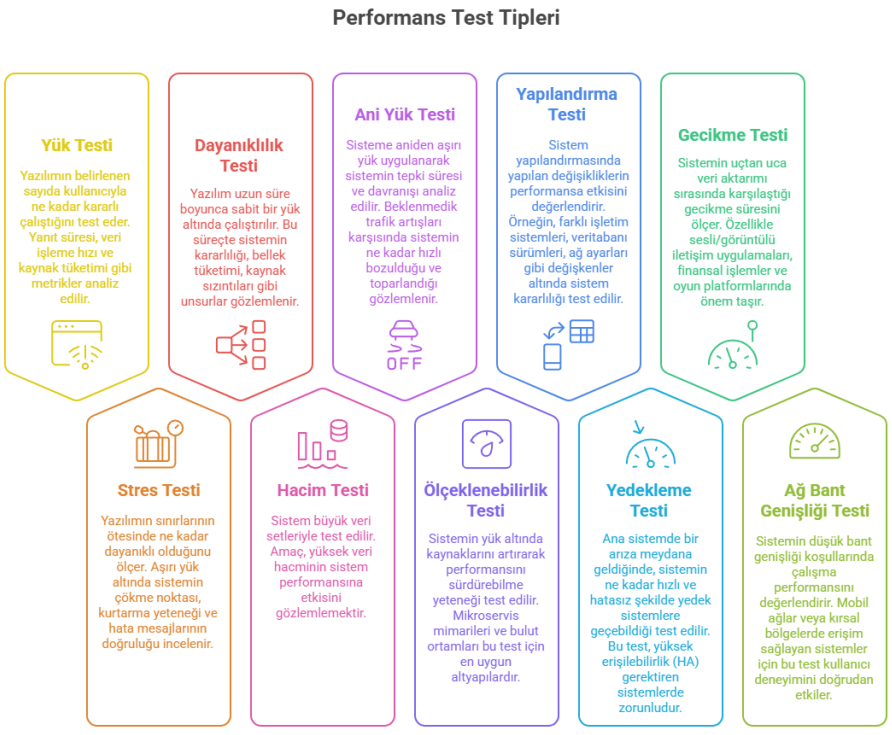
Performans Testi: Bir yazılım sisteminin, belirli koşullar altında, hız, yanıt süresi, kaynak kullanımı, kararlılık ve ölçeklenebilirlik gibi kriterler açısından nasıl çalıştığını analiz eden bir test türüdür. Yazılımın sadece doğru çalışıp çalışmadığını değil, aynı zamanda beklenen süre içerisinde çalışıp çalışmadığını da ortaya koyar. Bu nedenle performans testi, sistemin kullanıcı ihtiyaçlarını karşılayıp karşılamadığını değerlendirmek için kritik bir süreçtir [6], [10], [1]. Performans Testinin Amaçları: (a) Kapasite Belirleme: Yazılımın kaç eşzamanlı kullanıcıyı destekleyebileceğini belirlemek. (b) Yanıt Süresi Ölçümü: Belirli işlemlerin ne kadar sürede tamamlandığını tespit etmek. (c) Kararlılık Analizi: Uzun süreli kullanımda sistemin aynı performansı gösterip göstermediğini ölçmek. (d) Kaynak Kullanımı Değerlendirmesi: Sistem

kaynaklarının (CPU, RAM, ağ, disk) verimli kullanılıp kullanılmadığını analiz etmek. (e) Darboğazların Tespiti: Sistemin hangi bileşenlerinde performans düşüklüğü yaşandığını belirlemek.

Performans testleri, yazılım geliştirme sürecinin önemli bir parçasıdır çünkü yazılımın teknik olarak doğru çalışması, gerçek kullanım ortamında da başarılı olacağı anlamına gelmez. Özellikle yüksek trafik alan sistemlerde (örneğin bankacılık, e-ticaret, devlet portalları) performans sorunları müşteri kaybına, marka itibarının zedelenmesine ve finansal zararlara neden olabilir. Ayrıca kullanıcı deneyimi açısından da performans kritik öneme sahiptir. Örneğin, bir web sayfasındaki 2 saniyelik gecikme kullanıcıların %40'ının siteyi terk etmesine neden olabilir [11].

Performans testleri, yazılımın sadece çalışabilir olup olmadığını değil, aynı zamanda ne kadar verimli, ölçeklenebilir ve dayanıklı olduğunu da ölçer. Bu testler, sistemin mevcut yük altındaki davranışını anlamakla kalmaz, gelecekteki genişleme ihtiyaçlarını ve olası arızalara karşı dayanıklılığını da değerlendirir. Aşağıda, performans testlerinin alt türleri detaylandırılmıştır:

Performans testleri sistemlerin farklı ihtiyaçlarına yanıt verecek şekilde çeşitlenmiştir. Şekil 2.4, yazılımın kararlılık, tepki süresi, ölçeklenebilirlik ve sınır dayanımı gibi yönlerini ölçmek amacıyla uygulanan farklı performans test türlerini kapsamlı şekilde sınıflandırmaktadır. Aşağıda her bir test türü, amacı, uygulama zamanı ve örnekleriyle detaylandırılmıştır:



Şekil 2.4. Performans Test Tipleri

Yük Testi (Load Testing): Yazılımın belirlenen sayıda kullanıcıyla ne kadar kararlı çalıştığını test eder. Tipik olarak sistemin normalden yüksek olmayan yük altında ne kadar başarılı çalıştığını gözlemlemek için kullanılır. Yanıt süresi, veri işleme hızı ve kaynak tüketimi gibi metrikler analiz edilir. Örneğin, bir alışveriş sitesine aynı anda 1.000 kullanıcının giriş yapması simüle edilir [20].

Stres Testi (Stress Testing): Yazılımın sınırlarının ötesinde ne kadar dayanıklı olduğunu ölçer. Aşırı yük altında sistemin çökme noktası, kurtarma yeteneği ve hata mesajlarının doğruluğu incelenir. Bu test türü, felaket senaryolarına hazırlık için kritik önemdedir. Genellikle aşırı kullanıcı girişi, büyük veri akışı veya eşzamanlı işlemlerle uygulanır [18].

Dayanıklılık Testi (Soak Testing): Yazılım uzun süre boyunca sabit bir yük altında çalıştırılır. Bu süreçte sistemin kararlılığı, bellek tüketimi, kaynak sızıntıları (örneğin memory leak) gibi unsurlar gözlemlenir. Kritik kurumsal uygulamalar ve sürekli çalışan sunucular için vazgeçilmezdir [19].

Hacim Testi (Volume Testing): Sistem büyük veri setleriyle test edilir. Amaç, yüksek veri hacminin sistem performansına etkisini gözlemlemektir. Örneğin, 100 milyon satırlık bir veritabanında arama ve sıralama işlemleri yapılır. Bu test, özellikle büyük veri uygulamaları ve raporlama sistemlerinde önemlidir [19].

Ani Yük Testi (Spike Testing**):** Sisteme aniden aşırı yük uygulanarak sistemin tepki süresi ve davranışı analiz edilir. Beklenmedik trafik artışları karşısında sistemin ne kadar hızlı bozulduğu ve toparlandığı gözlemlenir. Özellikle viral kampanyalarda veya canlı yayın servislerinde sıkça uygulanır [18].

Ölçeklenebilirlik Testi (Scalability Testing): Sistemin yük altında kaynaklarını artırarak (örneğin CPU, bellek, sunucu) performansını sürdürebilme yeteneği test edilir. Mikroservis mimarileri ve bulut ortamları bu test için en uygun altyapılardır. Yatay ve dikey ölçekleme senaryoları incelenir [19].

Yapılandırma Testi (Configuration Testing): Sistem yapılandırmasında yapılan değişikliklerin performansa etkisini değerlendirir. Örneğin, farklı işletim sistemleri, veritabanı sürümleri, ağ ayarları gibi değişkenler altında sistem kararlılığı test edilir [18].

Yedekleme Testi (Failover Testing): Ana sistemde bir arıza meydana geldiğinde, sistemin ne kadar hızlı ve hatasız şekilde yedek sistemlere geçebildiği test edilir. Bu test, yüksek erişilebilirlik (HA) gerektiren sistemlerde zorunludur [17].

Gecikme Testi (Latency Testing**):** Sistemin uçtan uca veri aktarımı sırasında karşılaştığı gecikme süresini ölçer. Özellikle sesli/görüntülü iletişim uygulamaları, finansal işlemler ve oyun platformlarında önem taşır [16].

Ağ Bant Genişliği Testi (Bandwidth Testing): Sistemin düşük bant genişliği koşullarında çalışma performansını değerlendirir. Mobil ağlar veya kırsal bölgelerde erişim sağlayan sistemler için bu test kullanıcı deneyimini doğrudan etkiler [9]. Performans testleri yalnızca temel senaryolarla sınırlı değildir. Modern yazılım mimarilerinde farklı amaçlara hizmet eden çok sayıda özelleşmiş performans test türü vardır [16], [19]. Aşağıda en yaygın 10 tür detaylı şekilde açıklanmıştır.

Performans test süreci aşağıdaki adımlardan oluşmaktadır:

- **Hedeflerin Belirlenmesi:** Uygulamanın hedeflediği kullanıcı sayısı, işlem hacmi ve kabul edilebilir yanıt süreleri tanımlanır.
- **Test Ortamı Hazırlığı:** Gerçek ortama mümkün olduğunca yakın bir yapı oluşturulur. Sunucular, ağ yapısı, veri yükü vb. ayarlanır.
- **Senaryo Geliştirme:** Gerçek kullanıcı davranışları simüle edilerek oturum açma, arama, ödeme yapma gibi işlemler tanımlanır.
- **Araç Seçimi ve Uygulama:** Apache JMeter, LoadRunner, k6, Gatling gibi performans test araçları kullanılır.
- **Testin Uygulanması:** Belirlenen senaryolar belirli zaman aralıklarıyla, eşzamanlı kullanıcı sayılarıyla ve çeşitli veri setleriyle yürütülür.
- **Metrik İzleme:** CPU kullanımı, bellek kullanımı, istek başına yanıt süresi, başarısız istek oranı, ağ bant genişliği gibi metrikler izlenir.
- **Analiz ve Raporlama:** Toplanan verilerle grafiksel ve sayısal analizler yapılır. Darboğazlar belirlenir, iyileştirme önerileri sunulur.

Kullanılan başlıca araçlar aşağıda listelenmiştir.

- **Apache JMeter:** Açık kaynaklı, GUI tabanlı, Java ile geliştirilmiş bir araçtır.
- **LoadRunner:** Kurumsal düzeyde, protokol desteği çok geniş, ticari bir çözümdür.
- **k6:** JavaScript ile script yazılabilen, özellikle geliştirici dostu açık kaynaklı bir araçtır.
- **Gatling:** Scala temelli, senaryo yazımı güçlü, dağıtık test desteği olan bir araçtır.

Performans test metrikleri aşağıdaki gibi açıklanmıştır:

Yanıt Süresi (Response Time): Bir kullanıcının istekte bulunmasından, sistemin yanıt vermesine kadar geçen toplam süredir [5], [10]. Kullanıcı memnuniyetinin doğrudan bir göstergesidir. Kısa yanıt süreleri daha iyi bir kullanıcı deneyimi sağlar [5]. Veritabanı sorgularının optimize edilmesi, önbellekleme stratejilerinin uygulanması ve içerik dağıtım ağlarının (CDN) kullanılması konusunlarında yorum getirilmesini sağlar [16].

İşlem Hacmi (Throughput): Belirli bir zaman diliminde sistemin işleyebildiği işlem veya veri miktarıdır. Genellikle saniyede işlem sayısı (TPS) veya istek sayısı (RPS) olarak ölçülür [10], [16]. Sistemin kapasitesini ve verimliliğini gösterir. Yüksek işlem hacmi, sistemin daha fazla kullanıcıya hizmet verebilme yeteneğini ifade eder [5]. Yatay ölçekleme, yük dengeleme ve uygulama mantığının optimize edilmesi gibi iyileştirme yöntemleri konusunda yorum getirilmesini sağlar [16], [18].

Gecikme (Latency): Bir isteğin sistem tarafından alınmasından, ilk yanıtın gönderilmesine kadar geçen süredir [7], [15]. Gerçek zamanlı uygulamalarda düşük gecikme süreleri, kullanıcı deneyimini doğrudan etkiler [15]. Ağ optimizasyonu, veri sıkıştırma ve protokol iyileştirmeleri noktasında öneriler sunulmasına olanak tanır [18].

Hata Oranı (Error Rate): Toplam isteklere kıyasla başarısız olan isteklerin yüzdesidir [6], [25]. Sistemin güvenilirliğini ve kararlılığını değerlendirmek için kullanılır [6]. Hata ayıklama, istisna yönetimi ve kapsamlı test senaryolarının uygulanması noktasında yorum getirilmesini sağlar [25], [16].

Eşzamanlı Kullanıcı Sayısı (Concurrency): Aynı anda sistemi aktif olarak kullanan kullanıcıların sayısıdır [10], [25]. Sistemin eşzamanlı kullanıcı yükünü kaldırabilme kapasitesini gösterir [12]. Bağlantı havuzlama, kaynak yönetimi ve sistem mimarisinin optimize edilmesi konularında yorum getirilmesini sağlar [16].

CPU ve Bellek Kullanımı (CPU/Memory Utilization): Sistem kaynaklarının (işlemci ve bellek) kullanım düzeyidir [11]. Kaynakların verimli kullanımı, sistem performansı ve maliyet etkinliği açısından kritiktir [10]. Kod optimizasyonu, gereksiz işlemlerin azaltılması ve kaynak izleme araçlarının kullanılması konularında yorum getirilmesini sağlar [16], [17].

İlk Bayt Süresi (Time to First Byte - TTFB): Bir istemcinin istekte bulunmasından, sunucunun ilk baytı göndermesine kadar geçen süredir [20]. Web uygulamalarında sayfa yükleme süresinin önemli bir göstergesidir [10], [20]. Sunucu yanıt sürelerinin optimize edilmesi ve önbellekleme stratejileri konularında iyileştirme yapılması için öneriler sunar[16].

Sayfa Yükleme Süresi (Page Load Time): Bir web sayfasının yüklenmesi için geçen toplam süredir [16]. Kullanıcı deneyimi ve SEO açısından önemlidir [12]. Görsel ve script optimizasyonu, içerik dağıtım ağlarının kullanılmasında öneriler sunar[16], [18].

Bant Genişliği Kullanımı (Bandwidth Utilization): Sistemin mevcut ağ bant genişliğini ne kadar verimli kullandığını gösterir [16]. Ağ performansı ve veri iletim verimliliği açısından kritiktir [15]. Veri sıkıştırma, gereksiz veri iletiminin azaltılması ve CDN kullanımı noktasında öneriler sunulmasına olanak tanır [18].

İşlem Başarı Oranı (Transaction Success Rate): Başarılı tamamlanan işlemlerin, toplam işlemlere oranıdır [17]. Sistemin güvenilirliğini ve kullanıcı memnuniyetini değerlendirmek için kullanılır [10]. Hata yönetimi, işlem izleme ve kullanıcı geri bildirimlerinin analizi noktasında iyileştirme yapılmasına olanak tanır [16].

90./95./99. Yüzdelik Yanıt Süreleri (Percentile Response Times): Belirli bir yüzde dilimindeki isteklerin yanıt sürelerini gösterir [16], [18]. Kullanıcıların büyük çoğunluğunun deneyimini yansıtır ve uç durumları analiz etmek için kullanılır [10]. Performans darboğazlarının tespiti ve optimizasyonu konusunda fikir verir [16].

İşlem Süresi Dağılımı (Transaction Time Distribution): Farklı işlem türlerinin tamamlanma sürelerinin dağılımını gösterir [10]. Performans tutarsızlıklarını ve potansiyel sorunlu işlemleri belirlemek için kullanılır [17]. İşlem bazlı optimizasyon ve kaynak tahsisi konusunda fikir verir [18].

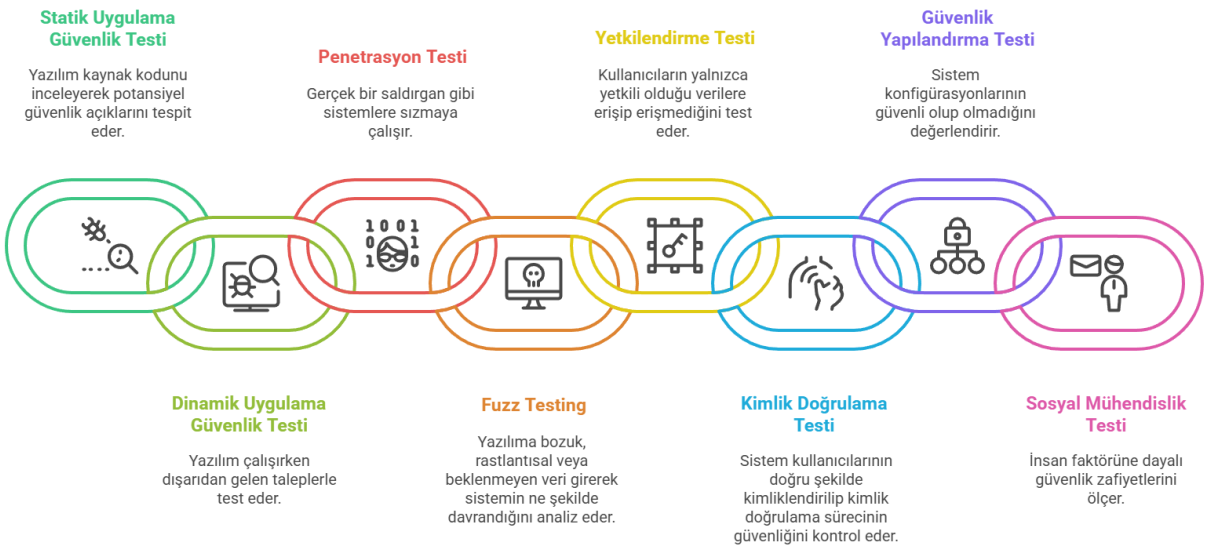
Kaynak Kullanım Eğilimleri (Resource Utilization Trends): Zaman içinde CPU, bellek, disk ve ağ kullanımındaki değişimleri gösterir [17]. Sistem kaynaklarının sürdürülebilirliğini ve potansiyel sızıntıları tespit etmek için kullanılır [10]. Kaynak izleme araçlarının kullanılması ve proaktif bakım stratejileri noktasında fikir verir [16].

Çalışma Süresi (Uptime): Sistemin kesintisiz olarak çalıştığı toplam süredir [25]. Hizmet sürekliliği ve güvenilirlik açısından kritiktir [10]. Yedekleme sistemleri, hata toleranslı mimariler ve düzenli bakım talimatları konusunda bilgi verir [16], [17].

Apdex Skoru (Application Performance Index**):** Kullanıcı memnuniyetini ölçen bir metriktir. Belirli bir yanıt süresi eşiğine göre, kullanıcı deneyimini "memnun", "toleranslı" ve "memnuniyetsiz" olarak sınıflandırır [19]. Kullanıcı deneyiminin nicel bir göstergesidir [15]. Performans iyileştirmeleri ve kullanıcı geri bildirimlerinin analizi konusunda iyileştirme yöntemleri sunar [18].

İş Değeri ve Katkısı: Performans testi, sistemin sadece çalışmasını değil, beklentiler dahilinde ve sürdürülebilir çalışmasını güvence altına alır. Müşteri kaybını, marka itibarını ve veri merkezine binen maliyeti azaltır. Günümüzde kullanıcıların %70'ten fazlası, yavaş açılan bir uygulamayı terk ettiğini ifade etmektedir [22]. Dolayısıyla performans testleri, sadece teknik ekiplerin değil, işletme stratejisinin de bir parçası olmalıdır.

Güvenlik Testi: Güvenlik testi, yazılımın gizlilik, bütünlük, erişilebilirlik ve güvenlik açıklarına karşı dayanıklılığını test eden kritik bir test türüdür. Modern yazılımlarda sadece temel doğrulama ve yetkilendirme değil, aynı zamanda veri koruma, sistem sınırlarının güvenliği ve dış tehditlere karşı koruma test edilmelidir.



Şekil 2.5. Güvenlik Testi Çeşitleri ve Kısa Tanımları

Şekil 2.5'de Güvenlik test türleri gösterilmiştir. Aşağıda bu test türleri detaylı şekilde sunulmuştur:

- Statik Uygulama Güvenlik Testi (SAST): Yazılım kaynak kodu incelenerek potansiyel güvenlik açıkları tespit edilir. Kod analizi sırasında kod çalıştırılmaz. Erken aşamada güvenlik hataları bulunabilir. Genellikle IDE eklentileri, CI/CD süreçleri ve kod tarayıcı araçlarıyla uygulanır. Kullanılan Araçlar: SonarQube, Fortify Static Code Analyzer, Checkmarx [24].
- Dinamik Uygulama Güvenlik Testi (DAST): Yazılım çalışırken, gelen taleplerle test edilir. Web uygulamalarında yaygındır. Gerçek kullanıcı davranışları simüle eder. SQL Injection, XSS gibi açıklar aranır. OWASP ZAP, Burp Suite, Acunetix araçları kullanılır [25].

- Penetrasyon Testi (Pen Test): Gerçek bir saldırgan gibi sistemlere sızmaya çalışılır. Sistemler, ağlar ve uygulamalar üzerindeki zafiyetlerin istismar edilebilirliği test edilir. En gerçekçi senaryoları sağlar [25]. Testler sırasında Metasploit, Kali Linux, Nessus, Nmap araçları kullanılır [26].
- Fuzz Testing: Yazılıma bozuk, rastlantısal veya beklenmeyen veri girilerek sistemin ne şekilde davrandığı analiz edilir. Bu test, bellek taşmaları, çökme ve güvenlik açıklarını ortaya çıkarabilir [27].
- Yetkilendirme Testi: Kullanıcıların yalnızca yetkili olduğu verilere erişip erişmediği test edilir. Rol bazlı erişim kontrolleri doğrulanır [24].
- Kimlik Doğrulama Testi: Sistem kullanıcılarının doğru şekilde kimliklendirilip kimlik doğrulama sürecinin güvenliği kontrol edilir. Çok faktörlü kimlik doğrulama (MFA) senaryoları da test edilir [24].
- Güvenlik Yapılandırma Testi: Sistem konfigürasyonlarının (örneğin güvenlik duvarı, port yönetimi, TLS/SSL ayarları) güvenli olup olmadığı değerlendirilir [26]. Testler sırasında CIS-CAT (CIS Benchmark), Lynis, OpenVAS araçlar kullanılır [26].
- Sosyal Mühendislik Testi: İnsan faktörüne dayalı güvenlik zafiyetlerini ölçer. Phishing, sahte aramalar ve bilinçsiz veri paylaşımı gibi yollarla sistemin güvenliği test edilir. Phishing simülasyonu, farkındalık eğitimi sonrası test, sahte e-posta senaryoları gibi test profilleri uygulanır [28].
- Kullanılabilirlik Testi (Usability Testing): Yazılımın son kullanıcılar tarafından ne derece kolay, anlaşılır ve verimli bir şekilde kullanılabildiğini değerlendirir [21], [11]. Arayüz düzeni, kullanıcı rehberliği, hata mesajlarının açıklığı, görev tamamlama süresi, tıklama sayısı ve öğrenme eğrisi gibi faktörler ölçülür. Kullanıcıların yazılımı terk etmesine neden olabilecek karmaşık arayüzler, zayıf yönlendirmeler veya düşük görsel hiyerarşiler gibi problemler tespit edilir. Doğrudan müşteri memnuniyetine etki eder. Yazılımın başarısı yalnızca teknik işlevlerine değil, kullanıcıyla kurduğu etkileşimin kalitesine de bağlıdır. Bu nedenle kullanılabilirlik testi, özellikle B2C uygulamalarda stratejik önemdedir.
- Uyumluluk Testi (Compatibility Testing): Yazılımın farklı platformlar, tarayıcılar, cihazlar, ekran boyutları, işletim sistemleri ve ağ ortamlarında tutarlı şekilde çalışıp çalışmadığını test eder [7], [10]. Mobil uyumluluk, çapraz tarayıcı uyumluluğu (Chrome, Firefox, Safari), donanımsal uyumluluk (Android/iOS cihazlar) ve farklı işletim sistemleri (Windows/Linux/macOS) gibi birçok varyasyon içerir. Yazılımın hedef kitlesi genişlediği

ölçüde, çeşitli ortamlarla sorunsuz çalışması gerekir. Bu test, evrensel kullanılabilirlik hedefi için kritiktir. Özellikle global kullanıcı kitlesi hedefleyen uygulamalarda pazara girişte rekabet avantajı sağlar. Uyumluluk eksikliği müşteri kaybına neden olabilir.

- Kurulum Testi (Installation Testing): Yazılımın yüklenme, güncellenme ve kaldırılma süreçlerinin hatasız işleyip işlemediğini değerlendirir [11], [22]. Yanlış yapılandırmalar, eksik bağımlılıklar, veri kaybı veya sistem çökmesi gibi kurulum kaynaklı hataların önüne geçilir. Son kullanıcı açısından yazılımın ilk izlenimini oluşturan kritik noktadır. İlk kurulumda yaşanan sorunlar, kullanıcıyı tamamen kaybetmeye neden olabilir.
- Taşınabilirlik Testi (Portability Testing): Yazılımın farklı ortamlarda çalıştırılabilirliğini ve başka sistemlere entegre edilebilirliğini test eder [7], [12]. Yedekleme sistemleri, konteyner ortamları (Docker), sanallaştırma teknolojileri (VMware), işletim sistemi geçişleri (Linux → Windows) gibi süreçlerde yazılımın davranışlarını izler. Yazılımı yeniden inşa etmeden veya büyük yapılandırma değişikliklerine gitmeden farklı ortamlarda çalıştırabilme yeteneği kazandırır. Özellikle SaaS modelinde geliştirilen uygulamalar için taşınabilirlik, pazarlanabilirlik ve operasyonel maliyetleri doğrudan etkileyen bir ölçüttür [7], [12].
- Erişilebilirlik Testi (Accessibility Testing): Engelli bireylerin yazılımdan eşit seviyede faydalanabilmesini sağlar [21], [12]. WCAG (Web Content Accessibility Guidelines) standartlarına uygunluk ölçülür. Görme engelliler için ekran okuyucu desteği, klavye ile tam gezinme, renk körlüğüne uygun kontrast ayarları gibi özelliklerin kontrolünü sağlar. Kamusal hizmet sunan web uygulamalarında erişilebilirlik yasal zorunluluktur. Aynı zamanda sosyal sorumluluğun ve etik geliştirmenin bir göstergesidir.

2.2 Literatür İncelemesi

Yazılım performans testleri, sistemlerin işlevselliği kadar ölçeklenebilirlik, yanıt süresi, sistem yükü altındaki davranış gibi kalite özelliklerini de değerlendirmek için kritik bir aşamadır. Ancak bu süreç, geleneksel yöntemlerle yürütüldüğünde zaman alıcı, hataya açık ve teknik bilgi gerektiren bir yapı sunar. Son yıllarda ortaya çıkan üretken yapay zekâ (GenAI) ve büyük dil modelleri (LLM), bu süreci otomatikleştirme ve kolaylaştırma konusunda güçlü bir alternatif olarak değerlendirilmektedir. Bu literatür tarama bölümünde, GenAI teknolojilerinin yazılım testi ve özellikle performans testi üzerindeki etkisi kapsamlı şekilde ele alınmıştır.

2.2.1 Test Otomasyonuna Yapay Zeka Yaklaşımları

Yazılım testi süreçlerinde yapay zekânın kullanımı son yıllarda hızla artmıştır. Literatürde, özellikle makine öğrenimi ve doğal dil işleme (NLP) teknikleri ile birim test, kabul testi ve entegrasyon testleri gibi alanlarda yüksek başarı elde edildiği görülmektedir. Jemai ve arkadaşları (2021), makine öğrenimi algoritmalarıyla tweet'lerin olumlu ve olumsuz olarak sınıflandırıldığı bir duygu analizi çalışmasında %99'a varan doğruluk oranlarına ulaşmışlardır. Bu çalışma, denetimli öğrenme algoritmalarının test verisi sınıflandırma gibi görevlerde kullanılabileceğini göstermiştir [29].

Benzer şekilde Sherifi ve arkadaşları (2024), büyük dil modellerini kullanarak yazılım test süreçlerinin otomatikleştirilmesi üzerine çalışmışlardır. Çalışmalarında birim test üretimi, çağrı grafiği analizi, test çalıştırma ve sonuç raporlama gibi adımların tamamen LLM ajanlar tarafından yürütülebildiğini göstermişlerdir. Bu çalışmanın en önemli katkısı, geliştiricinin doğal dilde yazdığı açıklamaların otomatik olarak test kodlarına dönüştürülebilmesidir [30].

Bhatia ve arkadaşları (2023), ChatGPT'yi Python programlarında birim test üretmek için kullanmışlardır. Deneylerinde, farklı prompt stratejileri ile oluşturulan testlerin, manuel yazılan testlere yakın doğruluk ve kapsama sunduğu görülmüştür. Ayrıca ChatGPT'nin yeniden kullanılabilir test kalıpları üretebilmesi sayesinde test sürecinin hızlandığı belirtilmiştir [31].

2.2.2 Performans Testleri ve Üretken Yapay Zekâ

Performans testleri, sistemin çoklu kullanıcı senaryoları altında nasıl davrandığını değerlendirmeye yönelik testlerdir ve genellikle Apache JMeter, Gatling, LoadRunner gibi araçlarla yürütülmektedir. Ancak bu araçların kullanımı belirli teknik bilgi ve manuel yapılandırmalar gerektirir. Literatürde bu sürecin otomatikleştirilmesi üzerine çalışmalar sınırlıdır fakat artan bir ilgi gözlemlenmektedir.

"Automating a Complete Software Test Process Using LLMs" başlıklı çalışmada, üretken yapay zekâ ile API testlerinin tamamen otomatik olarak oluşturulması ve yürütülmesi üzerine bir prototip geliştirilmiştir. Bu çalışma, performans testleri dahil olmak üzere test döngüsünün uçtan uca otomasyonunu hedefleyen nadir örneklerden biridir [32].

Yine benzer bir şekilde, VALTEST isimli bir framework'te, LLM'ler tarafından otomatik olarak üretilen test senaryolarının doğruluğu, token olasılıkları ile değerlendirilmiştir. Bu sistem, özellikle karmaşık test girdilerinin doğrulanmasında etkili bir yöntem sunmaktadır [33].

Ancak doğrudan performans testlerini hedef alan akademik çalışmaların sayısı halen sınırlıdır. Bu bağlamda bu tez çalışması, üretken yapay zekânın performans testi alanındaki potansiyelini ilk kez bütünsel olarak değerlendirmeyi amaçlamaktadır.

2.2.3 Gradio, GPT ve LocalLLM ile Test Otomasyonu

Literatürde üretken yapay zekânın test süreçlerine entegrasyonu, sadece arka uç algoritmaları ile sınırlı değildir. Kullanıcı arayüzleri de test otomasyonunun etkinliğini doğrudan etkilemektedir. Bu amaçla Gradio gibi açık kaynak kütüphaneler, kullanıcıların yapay zekâ ile etkileşimini kolaylaştırmak adına yaygın olarak kullanılmaktadır. Gradio arayüzleri ile, kullanıcıdan alınan kısa bir prompt yardımıyla test senaryosu oluşturmak, veri seti üretmek ve test sonuçlarını analiz etmek mümkün hâle gelmektedir.

GPT-3.5, GPT-4 gibi modellerin yanı sıra, veri gizliliği ve çevrimdışı kullanım ihtiyacına yönelik olarak geliştirilen Local Large Language Model çözümleri de literatürde dikkat çekmektedir. Yerel modeller, özellikle kurumsal ve güvenlik odaklı uygulamalarda tercih edilmekte ve herhangi bir dış API bağımlılığı olmadan çalışabilmektedir. Bu sayede sistemin tamamı kendi içinde, uçtan uca bir otomasyon döngüsü sunmaktadır.

2.2.4 Endüstriyel Yaklaşımlar ve Uygulama Farklılıkları

Yazılım test otomasyonuna yönelik üretken yapay zekâ çözümleri henüz endüstride tam anlamıyla yaygınlaşmamış olsa da, bazı araçlar bu yönde prototipler sunmaktadır. Microsoft'un GitHub Copilot aracı, yazılım geliştirme ortamında test üretimi, hata düzeltme ve refactoring gibi işlemleri büyük ölçüde otomatize etmektedir. Ancak bu tür araçların performans testi üretme ve yürütme konusunda yeterli olgunluğa ulaşmadığı literatürde sıkça vurgulanmaktadır.

Karhu ve arkadaşlarının yaptığı bir çalışma, endüstride üretken yapay zekâ destekli test araçlarının kullanılabilirliğini analiz etmiş ve bu teknolojilerin hâlâ deneysellik aşamasında olduğunu ortaya koymuştur [34].

2.2.5 Literatürdeki Boşluk ve Bu Tezin Katkısı

Yazılım test otomasyonu alanında yürütülen çalışmalar, özellikle birim test üretimi ve doğal dilden test kodu üretimi gibi alanlarda yoğunlaşmıştır. Ancak bu tez çalışması, bu eğilimin ötesine geçerek **yük testi (performans testi) senaryolarının üretimi ve analizine** odaklanarak alanyazındaki belirgin bir boşluğu doldurmayı hedeflemiştir.

Yapılan literatür değerlendirmesi sonucunda, genellikle birim test odaklı çalışmaların baskın olduğu, performans testi üretimi ve değerlendirmesi gibi daha karmaşık, çok bileşenli süreçlerin

yeterince araştırılmadığı gözlemlenmiştir. Örneğin Sherifi ve ark. (2024), test üretimini büyük dil modelleri aracılığıyla başarıyla otomatize etmiş olsa da; bu süreçte çoğunlukla statik yapıların test edilmesi hedeflenmiş ve JMeter gibi dinamik yük test platformlarıyla entegrasyon sağlanmamıştır [29]. Benzer şekilde Bhatia ve ark. (2023), ChatGPT tabanlı birim test üretiminde anlamlı doğruluk oranları yakalamış, ancak JMX ya da JTL gibi yapısal test dosyaları üzerine herhangi bir analiz sunmamıştır [30].

Diğer taraftan, "Automating a Complete Software Test Process Using LLMs" başlıklı çalışma, performans testleri de dâhil olmak üzere test döngüsünü uçtan uca ele almayı hedeflemiş; ancak çıktılarının detay seviyesi sınırlı kalmış ve JMeter gibi araçlara özgü yapıların üretimi ya da kaynak analizleri gibi metrikler detaylandırılmamıştır [32].

Bu tez ise hem **senaryo üretimi (JMX dosyaları)** hem de **test sonucu analizi (JTL dosyaları)** gibi iki katmanlı bir süreci değerlendirerek özgün bir yöntem geliştirmiştir. GPT-4, Claude 3, Gemini Pro gibi modern LLM'ler ile birlikte LLaMA 3, Mixtral ve Mistral gibi yerel modellerin farklı bağlamlarda nasıl davrandıkları deneysel olarak test edilmiş, sonuçlar hem **doğruluk (accuracy)** hem de **kavramsal tutarlılık (coherence)**, **yapısal derinlik (depth)** ve **kaynak verimliliği (RCI)** gibi metriklerle detaylandırılmıştır.

Buna ek olarak, bu çalışmada kullanılan değerlendirme stratejileri sadece çıktı kalitesi ile sınırlı kalmamış; aynı zamanda her bir modelin başarısını etkileyen arka plan faktörleri de (örneğin, prompt uzunluğu ile bağlamsal bütünlük, sentez yeteneği, eğitim veri kümesi, model mimarisi ve parametre yoğunluğu gibi) sistematik biçimde analiz edilmiştir.

Bu Tezin Katkısı

- Yük testine odaklanma: Literatürde birim testlerin ötesine geçilerek performans testlerinin ilk kez bu düzeyde otomatikleştirilmesi sağlanmıştır.
- JMeter entegrasyonu: .jmx ve .jtl gibi test araçlarının doğal formatları üzerinden üretim ve analiz gerçekleştirilmiş, bu dosyalar LLM'ler tarafından yönetilmiştir.
- Model mimarisi karşılaştırması: Hem bulut tabanlı (GPT-4, Claude 3, Gemini Pro) hem yerel (LLaMA 3, Mixtral, Mistral 7B) modeller değerlendirilerek veri gizliliği ve kaynak kullanımı bağlamında stratejik öneriler geliştirilmiştir.
- Uçtan uca otomasyon döngüsü: Gradio arayüzü ile kullanıcıdan gelen tek bir prompt üzerinden tüm sürecin başlatılması ve yürütülmesi sağlanarak pratik bir test platformu simülasyonu yapılmıştır.

- Analitik ve grafiksel çıktıların yapay zekâ ile üretilmesi: JTL analiz sürecinde, yapay zekâ modellerinin response time eğilimleri, RT90 anomalileri, latency grafikleri gibi yapıları yorumlayabilmesi değerlendirilmiştir.

Tablo 2.3.1 Literatür ile Karşılaştırma

Çalışma	Katkı	Eksik Kalan Yön	Bu Tezle İlişkisi
Jemai al. (2021)	ML ile metin sınıflandırma	Yalnızca duygu analizi odaklı	Veri etiketleme ve sınıflandırma temeli oluşturur
Sherifi al. (2024)	LLM ile otomatik test üretimi	Yalnızca birim test	Doğal dilden test üretimi yaklaşımına öncülük eder
Bhatia al. (2023)	ChatGPT ile Python testleri	Performans testi yok	LLM ile test kalıbı üretimini destekler
"Automating..." (2023)	LLM ile API test otomasyonu	Yalnızca basit testler, derinlemesine yapılandırma eksik	Uçtan uca süreç otomasyonunu gösterir ama sınırlı
VALTEST	Token olasılığı ile senaryo değerlendirme	Teknik metrik yetersiz	Değerlendirme mantığını geliştirir ama uygulama zayıf
Karhu al. (2022)	Endüstriyel uygulamaların değerlendirmesi	Deneysel sınırlılıklar	Gerçek ortamda LLM test uygulamalarına ışık tutar
Tez Çalışmamız	LLM ile performans testi üretimi ve analizi	-	Literatürdeki boşluğu sistematik ve deneysel olarak doldurur

Tablo 2.1'de, literatürde yer alan güncel çalışmalar LLM (Büyük Dil Modelleri) ve yazılım testi ilişkisi bağlamında karşılaştırılmıştır. Her bir çalışma, katkı sağladığı alan, eksik kalan yönleri ve bu tez çalışmasıyla olan ilişkisi açısından değerlendirilmiştir. Mevcut literatürde daha çok duygu analizi, birim testi veya basit otomasyon senaryolarına odaklanıldığı; performans testi üretimi ve analizi gibi daha kapsamlı ve derinlemesine yaklaşımların yeterince ele alınmadığı görülmektedir. Bu tez, bu eksikliği gidererek LLM'lerin performans testi bağlamındaki potansiyelini sistematik ve deneysel olarak ortaya koymayı hedeflemektedir.

3. YÖNTEM

Bu çalışmada yürütülen performans testlerinin temelini oluşturan sistemler, Softtech bünyesinde geliştirilen/kullanılan bankacılık servisleri ve üretim ortamında aktif olarak kullanılan iki temel mikroservis tabanlı yazılım ürünü olan **Proemtia** ve **Lokum** sistemleridir. Her iki sistem de modern yazılım mimarilerinin gereksinimlerine uygun olarak geliştirilen, yüksek ölçeklenebilirlik ve esneklik sunan mikroservis yapıları ile tasarlanmıştır. Bu mikroservisler, RESTful API mimarisine dayalı olup, her bir servis uç noktası Swagger/OpenAPI standartlarına uygun şekilde dökümante edilmiştir.

Proemtia platformu banka bünyesinde geliştirilen endistüriyel pazaryeri platformudur. Proemtia sistemi içinde verilerin toplanması, işlenmesi ve dış sistemlere entegrasyonu gibi işlemleri yöneten bir kurumsal entegrasyon platform yapısı ve mikroservis mimarisi bulunmaktadır.

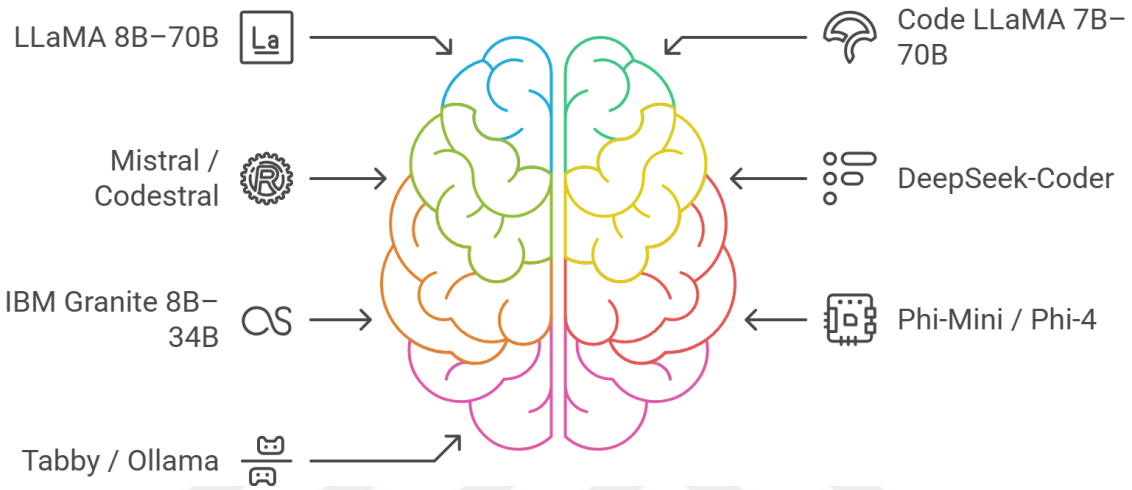
Lokum ise daha çok iç süreçleri ve sistemler arası operasyonel iletişimi destekleyen aynı zamanda şirket içi sosyal kültürün oluşturulması içinde kullanılan bir platform yer almaktadır. Bu platform bir servis mimarisine sahip aynı zamanda mobil APP ile de yaygınlaştırılan bir cross yapı içermektedir. Bu iki farklı yapı haricinde bankacılık sistemlerinde bulunan, müşteri verisi yönetimi, hesap hareketleri, işlem onay süreçleri, ödeme sistemleri ve güvenlik politikaları gibi kritik işlevleri gerçekleştiren servis kümeleri/mimarilerini de içermektedir.

Bu sistemlerin test süreçleri, gerçek dünyadaki kullanım senaryolarını yansıtacak şekilde yapılandırılmış; yüksek eş zamanlılık altında hizmet verecek şekilde kurgulanmıştır. Her bir servis, API uç noktaları üzerinden test edilecek şekilde ayrıştırılmış, fonksiyonel bağımlılıkları dikkate alınarak kapsamlı test koleksiyonları oluşturulmuştur. Böylece, sistemlerin HTTP isteklerine karşı verdikleri yanıt süreleri, hata toleransı ve kaynak kullanım düzeyleri gibi metriklerin ölçülmesi mümkün hale getirilmiştir.

Bu çalışmada yazılım performans testi süreçlerinde üretken yapay zekâ (ÜYZ) teknolojilerinin entegrasyonu detaylı olarak ele alınmıştır. Kullanılan sistemler arasında Proemtia, Lokum ve çeşitli bankacılık uygulamaları yer almakta olup, API tabanlı hizmetlerin analizine odaklanılmıştır. Test sürecinin planlanmasından yürütülmesine kadar tüm aşamalarda LLM (Large Language Models), GPT-4, GPT-4o, GPT-3.5, OpenAI API'leri ve bazı Local LLM çözümleri kullanılmıştır.

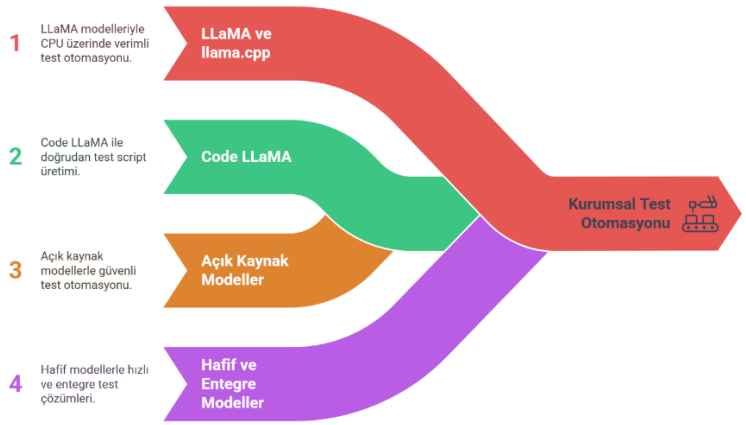
3.1 Yazılım Test Süreçlerinde Yerel (Local) LLM Çözümleri

Bulut tabanlı GPT servisleri yüksek üretkenlik sağlasa da, birçok kurum veri gizliliği, gecikme, maliyet ve güvenlik gereksinimleri nedeniyle yerel (on-premise) LLM kullanımına yönelmektedir. Yerel modeller, hassas test verilerinin kurum dışına çıkmasını engelleyerek yasalara uygunluk sağlar ve kurumsal test otomasyon süreçlerinde bağımsız yapay zekâ altyapısı kurulmasına olanak tanır. Şekil 3.1 'de yazılım test süreçlerindeki yerel LLM çözümleri verilmiştir.



Şekil 3.2. Yapay Zeka Modellerine Genel Bakış

Şekil 3.2, yerel büyük dil modellerinin farklı yetenekleri kullanılarak kurumsal test otomasyonuna nasıl entegre edilebileceğini bütüncül bir yaklaşımla ortaya koymaktadır.



Şekil 3.2. Yerel large Language Model ile Test Otomasyonu

LLaMA ve llama.cpp**:** (Large Language Model Meta AI), Meta AI tarafından açık kaynak sunulan güçlü bir model ailesidir. 2024'te yayımlanan **LLaMA 3** sürümleri (8B, 70B, 405B parametre) ile özellikle 128k token bağlam penceresine sahip modeller, yüksek performans

sağlamaktadır [37]. Bu modeller, llama.cpp aracılığıyla CPU veya düşük güçlü altyapılarda bile çalıştırılabilir hale gelmektedir [38]. llama.cpp, Minimal bağımlılık gerektiren bir C++ kütüphanesidir ve AVX, NEON gibi optimizasyonlarla CPU üzerinde bile verimli inference sağlar. Raspberry Pi gibi gömülü cihazlarda dahi çalışabileceği belirtilmektedir [39][40]. Bu mimari, test kodu üretimi, assertion tanımları, retry mekanizmaları gibi süreçler için kullanılabilir pratiklik sunar.

Uygulama Örneği**:** Kurumsal test pipelines'ında, llama.cpp tabanlı LLaMA 8B modeli offline ortamda JUnit ve JMeter senaryoları için assertion yapıları ve optimizasyon önerileri üretebilir.

- Code LLaMA – Kod Odaklı LLM: kod üretimi üzerine optimize edilmiş LLaMA 2 temelli modeller içerir (7B, 13B, 34B, 70B parametre) [41]. Özellikle Python varyantları, HumanEval ve MBPP benchmark'larında %65–67 gibi yüksek geçerli sonuçlar elde etmektedir [42],[43].
- Kullanım Durumu: Bir iş gereksiniminden doğrudan test script'i üretmek için prompt edilen eğitim programı ile, Code LLaMA test senaryoları, assertion'lar veya hata durumu yönetim kodları oluşturabilir.

3.1.1 Açık Kaynak Kod Modelleri: Mistral, DeepSeek-Coder ve Granite

Birçok açık kaynaklı kod üretim LLM'i, test otomasyonları için güçlü alternatifler sunmaktadır:

- Mistral 7B / Codestral 22B**:** Apache-2 lisanslı, hafif ama performanslı modellerdir [44].
- DeepSeek-Coder**:** 16k token bağlamıyla test-case üretimine ideal çözümler sunmaktadır [45].
- IBM Granite (8B–34B)**:** Benchmark sonuçlarına göre Code LLaMA'yı geçebilecek kod üretim kapasitesine sahiptir [54].

Bu modeller, veri gizliliği ve lisans gereksinimleri olan kurumsal ortamlarda güvenli test üretimi sağlar.

3.1.2 Hafif ve Entregre Modeller: Phi-Mini, Tabby, Ollama

- Phi-3 Mini / Phi-4: 3.8B–14B parametre aralığındaki bu modeller, hızlı inceleme ve düşük gecikme gerektiren gömülü sistemler için idealdir [46].
- Tabby: Docker tabanlı, IDE destekli bir kod yardımcısıdır ve test otomasyon sürecine entegre edilebilir [48].
- Ollama & AnythingLLM**:** Çok modellilik ve hızlı prototip ile yerel chat asistanları sunar; test promptlarını interaktif olarak destekler [49].

Şekil 3.3. Entegrasyon ve Uygulama Senaryoları,

Şekil 3.3'e göre, yapay zekâ modelleri; test durumu otomasyonu, hata yönetimi, kendini iyileştiren test akışları ve gömülü/CI ortamlarında test üretimi gibi farklı entegrasyon alanlarında yüksek doğruluk, esneklik ve hız avantajları sunarak yazılım test süreçlerinde önemli iyileştirmeler sağlamaktadır.

Tablo 3.1. Modellerin Avantaj ve Dezavantajları

Model / Araç	Boyut (Params)	Avantajlar	Dezavantajlar	Önerilen Kullanım Alanı
LLaMA 8B–70B	8B–70B	Güçlü bağlam, offline, CPU ile çalışır	Büyük model çok kaynak tüketir	Assertion/senaryo üretimi
Code LLaMA 7B–70B	7B–70B	Kod üretiminde yüksek doğruluk (%65 HumanEval)	Context yönetimi zor olabilir	Test kodu, retry mekanizması
Mistral / Codestral	7B–22B	Apache-2 lisanslı, lisanslı kuruluşlarda uygun	Benchmark verileri sınırlıdır	Kurumsal test otomasyon
DeepSeek-Coder	1.3B–33B	16k token, test-case üretiminde efektif	Dokümantasyon eksik olabilir	UI test, jenerasyon
IBM Granite 8B–34B	8B–34B	Code LLaMA benchmark'larında önde	Kaynak kullanımı yüksek	Güvenli kurumsal ortam
Phi-Mini / Phi-4	3.8B–14B	Hafif, hızlı, gömülü cihaz desteği	Kod kalitesi sınırlı	Gömülü/CI ortamlarında üretim
Tabby / Ollama	—	IDE entegrasyonu, kullanıcı etkileşimi	CLI kullanımı sınırlı	Interaktif test otomasyonu

Tablo 3.1'de, farklı parametre boyutlarına sahip yapay zekâ modellerinin avantaj ve dezavantajları karşılaştırılarak, her bir modelin en uygun olduğu test otomasyon kullanım alanları belirlenmiştir.

3.1.3 Performans, Doğrulama ve Benchmark Sonuçları

LLaMA 8B**,** CPU'da GPT-4 ile karşılaştırıldığında zaman-maliyet dengesi açısından avantaj sağlar (MLCommons MLPerf v4.0 Testleri) [1],[4].

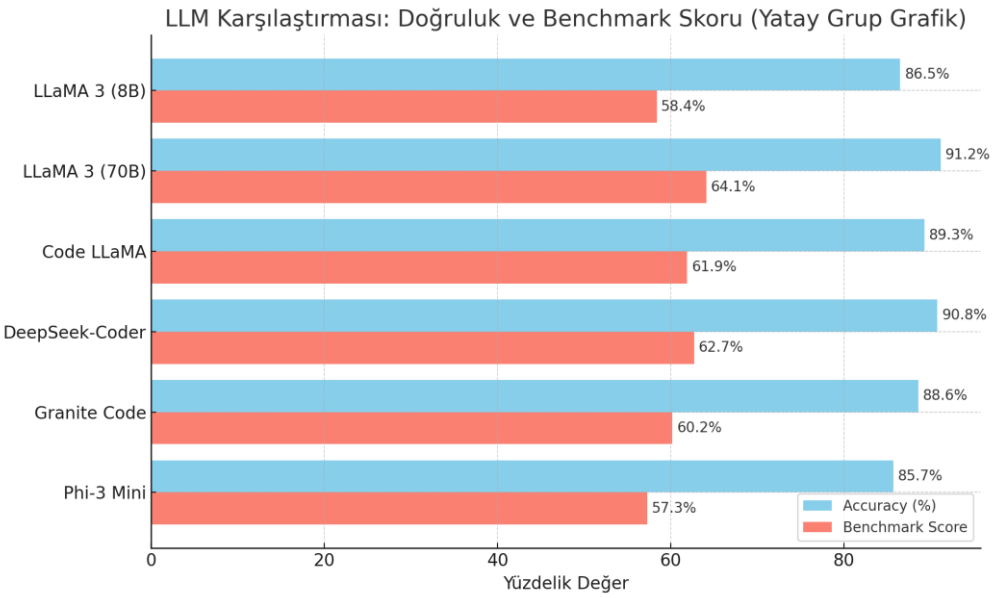
HumanEval benchmark'ında Code LLaMA 34B, MBPP'de %65–67 doğruluk değeri ile dikkat çekmiştir [43],[42],[50].

LLM ile test kodları üzerinde yapılan incelemeler: GPT-4o, LLaMA 3.3 ve Amazon Q gibi modellerden üretilen testlerin %60–90 oranında edilgen kalite, ancak %64 assertion hatası tespit edildi.

DeepSeek-Coder, test-case üretimi senaryoları için 16k bağlam penceresiyle ideal bir çözüm sunar [45].

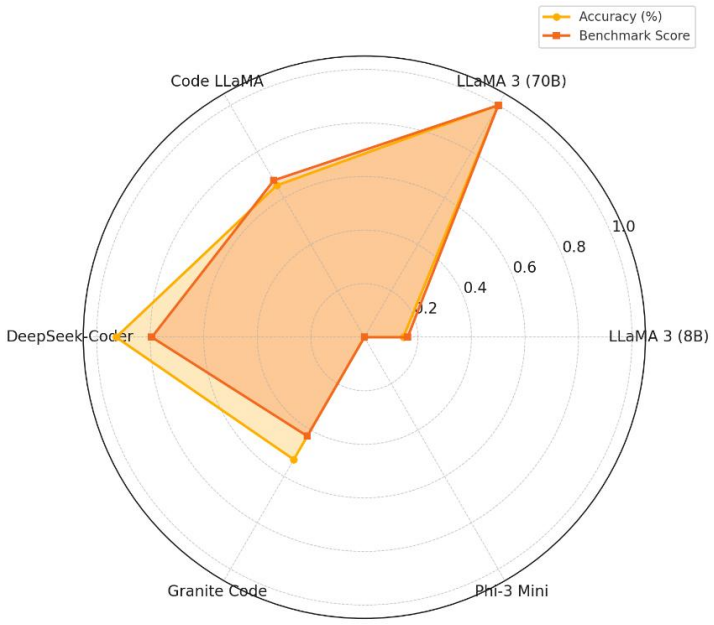
llama.cpp ile modeller GPU bağımsız, ARM-x86 gibi altyapılarda çalışabilirlik sağlar [38],[51].

Code LLaMA**,** benchmark'larda resmi GPT-4'e yakın performans gösterdi, %65–67 doğrulukla düşük maliyetli test üretimi sağlar.



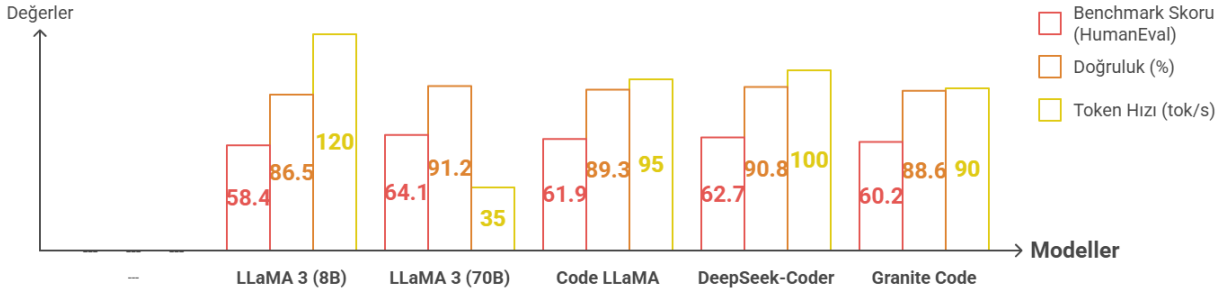
Şekil 3.4. LLM Karşılaştırma: Doğruluk ve Bencmark Skoru

Şekil 3.4'te, çeşitli büyük dil modellerinin doğruluk oranları ile benchmark skorları karşılaştırılarak test otomasyonundaki performans etkinlikleri nicel olarak değerlendirilmiştir.



Şekil 3.5. LLM Karşılaştırılması Radar Grafiği

Şekil 3.5'te, büyük dil modellerinin doğruluk ve benchmark performansları radar grafik ile görselleştirilerek, modeller arasındaki yetkinlik farkları bütüncül şekilde ortaya konmuştur.



Şekil 3.6. LLM Modellerinin Performans Karşılaştırması

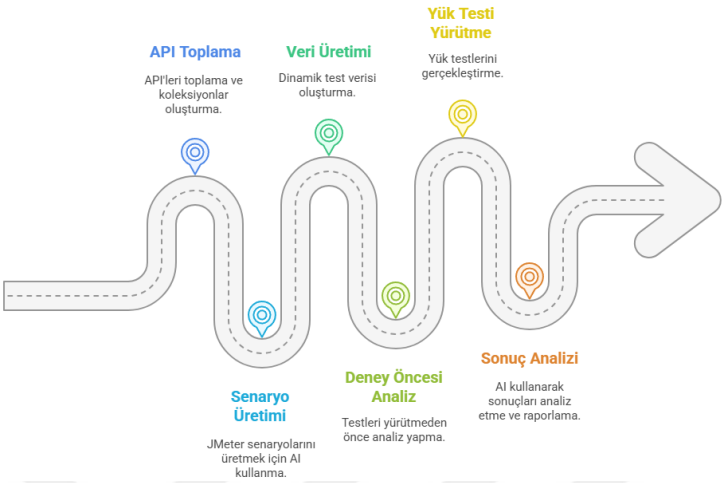
Şekil 3.6'da, büyük dil modellerinin doğruluk, benchmark skoru ve token üretim hızı açısından performans karşılaştırması yapılarak uygulama verimlilikleri nicel olarak değerlendirilmiştir.

Aşağıda deneysel süreçlerin her bir adımı ayrıntılı şekilde sunulmaktadır. Servisler üzerinde yapılan performans testleri**,** yalnızca testin yürütülmesiyle sınırlı kalmayıp şunları kapsar:

- API toplama ve koleksiyon oluşturma
- Yapay zeka ile JMeter senaryo üretimi
- Dinamik test verisi üretimi
- Deney öncesi analiz ve dry-run
- Yük testi yürütme
- AI tabanlı sonuç analizi ve raporlama

3.2 Yapay Zeka ile Perfomans Testi Süresince Gerçekleştirilen Adımlar

Burada amaç, yalnızca performans testi yapmak değil; sürecin her adımında LLM ve üretken yapay zekâyı kullanarak "self-healing" ve "prompt engineering" gibi yenilikçi yaklaşımları ve güvenilirliği test etmektir.



Şekil 3.7. Performans Testi Süreci

Şekil 3.7'de, yapay zekâ destekli performans testi süreci; API toplama, veri üretimi, senaryo oluşturma, deney öncesi analiz, yük testi yürütme ve sonuç analizi olmak üzere altı aşamada sistematik olarak sunulmuştur.

3.2.1 API Toplama ve Koleksiyon Oluşturma

Performans testi sürecinin temel adımlarından biri olan API toplama ve koleksiyon yapısı oluşturma, doğrudan testin kapsamını, doğruluğunu ve verimliliğini etkiler. Bu adımın nasıl gerçekleştirildiğini ve performans testine katkılarını akademik kaynaklarla inceleyelim:

Trafikten API Keşfi: Gerçek sistem trafiğinin incelenmesi, kullanılabilir tüm uç noktaların keşfinde etkilidir. TestGuild tarafından yapılan bir çalışmada belirtildiği gibi, API testlerinin erken aşamalarda sıklıkla çalıştırılması, performans darboğazlarının tespitini önemli ölçüde hızlandırır [52]. Bu nedenle, Postman Interceptor veya HTTP proxy kullanan yaklaşımlar, performans testlerinde kullanılacak tüm uç noktaları yalın bir şekilde ortaya çıkarır.

Swagger/OpenAPI Tabanlı Dönüşüm: Yaygın bir yöntem de yazılım üreticileri tarafından sağlanan OpenAPI/Swagger tanımlarının doğrudan Postman veya JMeter koleksiyonlarına dönüştürülmesidir. Bu, hem **test kapsamını genişletir** hem de yeni uç noktaların manuel belirlenmesi ihtiyacını ortadan kaldırır [53].

Çoklu Servis Zinciri (API Chaining): Mikroservis mimarisindeki uçtan uca test senaryoları oluşturmak için "API Chaining" yaklaşımı önemli rol oynar. Bir servisin cevabı diğerini tetiklediğinde, performans testi gerçekçi yük simülasyonu sağlar. Bu yöntemle test sırasında hizmetler arası etkileşimlerin doğruluğu ve performansı ölçülür [54].

Tablo 3.2. Çoklu Servis Zinciri (API Chaining)

Yaklaşım	Avantajı	Performans Testine Yansıması
Trafik bazlı koleksiyon	Gerçek dünya senaryolarını yakalar	Gerçekçi kaynak kullanım analizi sağlar
Swagger dönüşümü	Hızlı ve sistematik test kapsamı sağlar	Tüm uç noktalar önceden tanımlı, eksik kalmaz
API Zincirleme	Hizmetler arası bağımlılığı ortaya çıkar	Uçtan uca performans darboğazlarını tespit eder
LLM Tabanlı Otomatik Koleksiyon	Eksik parametreleri tamamlar, otomatik koleksiyon üretir.	Senaryo üretimi hızlanır, doğruluk artar.
El ile API Girişi ve Entegrasyonu	Yalın sistemlerde esneklik sunar, özel senaryolar oluşturulabilir.	Hedefli performans testi uygulanabilir, derinlemesine analiz yapılır.

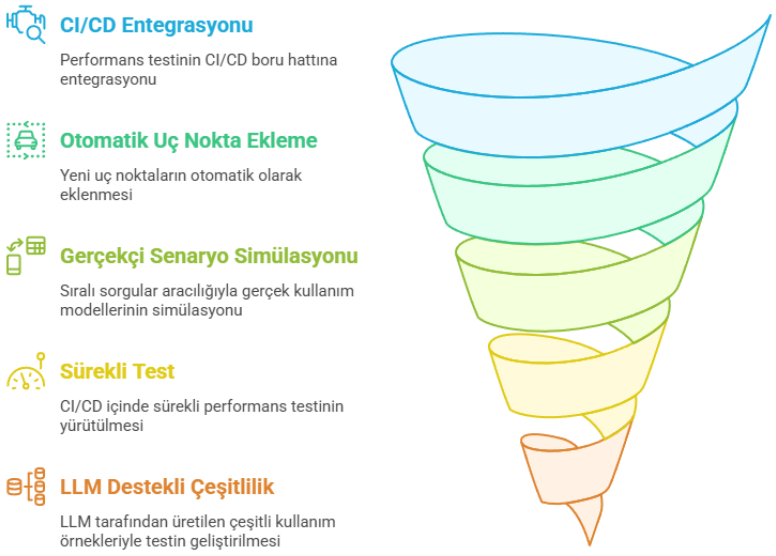
Tablo 3.2'de, farklı test koleksiyonu yaklaşımlarının avantajları ve performans testine olan yansımaları karşılaştırılarak; gerçekçilik, kapsamlılık, doğruluk ve esneklik açısından senaryo üretimine nasıl katkı sağladıkları ortaya konmuştur.

Otomasyon & LLM Tabanlı İyileştirme: Koleksiyon Oluşturma Otomasyonu: RapidAPI veya Swagger platformlarından çekilen uç noktalar, Postman SDK ya da LLM'ler yardımıyla koleksiyon haline dönüştürülmektedir. LLM altyapıları, eksik parametre tanımları ve JSON şemalarını otomatik tamamlayarak performans test senaryolarının kapsamını güçlendirir [55].

LLM Destekli API Test Senaryoları: Örneğin, "APITestGenie" modeli gerçek servis bilgileriyle Prompt'landığında %57'lik geçerlilik oranıyla çalışabilirken, tekrarlanan LLM çağrılarıyla bu oran %80'e kadar çıkabilmektedir[56]. Bu tarz yaklaşımlar, hem performans senaryolarında kullanılan veri çeşitliliğini artırır hem de test verisi üretim sürecini hızlandırır.

CI/CD Entegrasyonu ve Performans Takvimi: API koleksiyonu toplandıktan sonra, performans testinin CI/CD boru hattına entegrasyonu kritik hale gelir. Grafana k6 gibi araçlar performans testlerini düzenli bir şekilde tetiklerken, otomasyon sayesinde yeni veya değişen uç noktalar anında koleksiyonlara eklenir ve test edilebilir hale gelir [57].

Şekil 3.8'de, uçtan uca API performans testi süreci; CI/CD entegrasyonundan otomatik uç nokta eklemeye, gerçek senaryo simülasyonundan sürekli teste ve LLM destekli çeşitliliğe kadar adım adım bir yaklaşımla bütüncül olarak sunulmuştur.



Şekil 3.8. Uçtan Uca API Performans Testi Süreci

Performans Testine Etkileri

- Kapsam genişliği**:** Eksik uç nokta bırakılmaz; performans analizi her API için yapılır.
- Gerçekçi senaryolar**:** Sıralı sorgular sonucunda gerçek kullanım modeli simüle edilmiş olur.
- Süreklilik: CI/CD içinde sürekli test çalıştırması, performans regresyonunu önler [57][52].
- LLM destekli çeşitlilik**:** Otomatik veri üretimi sayesinde farklı kullanım örnekleriyle performans testi daha sağlam hale gelir [56].

3.2.2 Yapay Zeka ile JMeter Senaryo Üretimi ve Test Edilecek Sistemlerin Hazırlığı

API Koleksiyonu ve Test Senaryolarının Oluşturulması: Yazılım test otomasyon süreçlerinde senaryo üretiminin doğruluk, kapsayıcılık ve hız açısından kritik bir rolü vardır. Özellikle mikroservis mimarilerinde, API sayısının ve bağımlılıklarının artması, manuel senaryo oluşturmayı hem zaman alıcı hem de hataya açık bir sürece dönüştürmektedir. Bu noktada, üretken yapay zekâ (GenAI) tabanlı sistemler, performans testi senaryolarının otomatikleştirilmesinde devrim niteliğinde yenilikler sunmaktadır.

Özellikle OpenAI'nin GPT-4, Meta'nın Code LLaMA ve HuggingFace tabanlı modelleri gibi büyük dil modelleri (LLM'ler), kullanıcıdan alınan doğal dil girdileri doğrultusunda JMeter senaryolarını XML veya DSL (Domain Specific Language) formatında otomatik olarak oluşturabilmektedir [53], [55]. Bu sistemler, kullanıcı tarafından sağlanan bir Postman collection dosyasını veya Swagger/OpenAPI tanımını alarak, her bir endpoint için uygun

Thread Group, HTTP Sampler, PreProcessor, Assertion, Listener ve Timer gibi bileşenleri yapılandırılmış biçimde üretir [56].

Bu süreçte LLM modelleri aşağıdaki işlevleri üstlenmektedir:

API Tanıma ve Dönüşüm: JSON formatındaki Postman collection ya da Swagger spesifikasyonu doğal dil düzeyinde analiz edilerek test senaryosuna dönüştürülür.

Bağımlılık Algılama: Response içinde yer alan verilerin bir sonraki API çağrısında kullanılması gibi senaryolarda otomatik olarak JSON Extractor veya Regular Expression Extractor bileşenleri eklenir [54].

Senaryo Farklılaştırma: JMeter senaryoları sadece "tekil servis testi" değil, "uçtan uca senaryo zincirleme (chained requests)" mantığıyla yapılandırılabilir. Bu, mikroservislerin entegrasyon testlerini simüle etmek için elzemdir [52].

Test Verisi Üretimi: CSV Data Set Config bileşeniyle senaryoya entegre edilebilecek şekilde örnek veriler üretilir veya yapay olarak türetilir. Bu sayede veri çeşitliliği ve gerçeklik düzeyi artırılır.

Dinamik Yapılandırmalar: Prompt tabanlı model yönlendirmesiyle Number of Threads, Ramp-Up, Loop Count gibi parametreler kullanıcıya sorularak ya da öğrenilmiş kalıplarla tahmin edilerek belirlenebilir [57].

Yapay zekâ ile senaryo üretimi, sadece manuel iş yükünü azaltmakla kalmaz; aynı zamanda senaryo üretiminde tutarlılığı artırır ve hata oranını minimize eder. Özellikle yazılım projelerinin sık sık değişen API uç noktalarına sahip olduğu durumlarda, yapay zekâ bu değişimleri hızlıca işleyip güncel test senaryoları oluşturabilir [58].

Tablo 3.3. Yapay Zeka ile JMeter Senaryo Üretim Süreci

Adım	Açıklama	Yapay Zeka İşlevi
Girdi Tanıma	Postman collection veya Swagger/OpenAPI dokümanı model girdisi olarak alınır.	Veri yapısının NLP ile tanımlanması
API Tanımlama	Endpoint, metod, parametre, header gibi öğeler doğal dilde analiz edilerek test edilecek API'ler belirlenir.	JSON/XML analizi, kod önerisi
Servis Bağımlılığı Analizi	Servisler arası veri bağımlılığı tespit edilerek gerekli Extractor yapılandırmaları yapılır.	Bağımlılık örüntüsü çıkarımı
JMeter Bileşeni Üretimi	Thread Group, HTTP Sampler, Assertions, Pre/PostProcessors, Timers gibi test elemanları otomatik olarak oluşturulur.	Kod oluşturma, bileşen eşleme
Test Verisi Entegrasyonu	CSV Data Set Config kullanılarak test girdisi çeşitliliği artırılır.	Veri üretimi, varyasyon planlaması
Parametre Dinamikliği	Kullanıcı girdisine ya da örüntü çıkarımına göre thread, loop, süre gibi değerler belirlenir.	Prompt analizine dayalı tahminleme
Senaryo Optimizasyonu	Senaryo zincirleri oluşturularak uçtan uca (end-to-end) test akışı sağlanır, zamanlamalar optimize edilir.	LLM yönlendirmesiyle senaryo biçimleme

Tablo 3.3'te, yapay zekâ destekli JMeter senaryo üretim süreci adım adım açıklanarak, veri analizinden bileşen oluşturmaya ve senaryo optimizasyonuna kadar otomasyonun sağladığı katkılar ortaya konmuştur.

Özetle, üretken yapay zekâ destekli JMeter senaryo üretimi, test mühendisliğinde verimliliği artırmakta ve insan faktörüne dayalı kısıtları ortadan kaldırmaktadır. Bu uygulama, performans test süreçlerinin hem teknik doğruluğunu hem de operasyonel hızını ciddi şekilde geliştirmektedir.

Bunun için şu türde promptlar verilmiştir:

Lütfen aşağıdaki adımları takip ederek bir JMeter (.jmx) test planı üretin:

'''''Giriş Verileri:

Kullanıcıdan ya bir **Postman Collection JSON** dosyası ya da doğrudan bir **Swagger/OpenAPI** dokümantasyon bağlantısı (URL) sağlanacaktır.

Eğer Swagger linki mevcutsa:

API uç noktalarını, HTTP metotlarını (GET, POST, PUT, DELETE vb.), URL parametrelerini, gövde (body) yapılarını ve başlık (header) bilgilerini otomatik olarak parse edin.

Swagger üzerinden oluşturulan bilgileri içeren geçerli bir Postman Collection JSON formatı üretin.

Eğer Swagger linki yoksa:

Kullanıcıdan servis adı, metod tipi, endpoint URL'si, parametre tanımları, request body ve response örneği gibi tüm detaylar manuel olarak alınarak bir Collection üretin.

Üretilen collection, JMeter yapısına uygun bir şekilde dönüşüme tabi tutulacaktır.

Collection İçeriği ve API Akışı

Pre/Post-Script Analizi:

Collection içinde yer alan pre-request ve test (post-response) script'leri varsa, bunları uygun biçimde analiz edin.

Bu script'leri aşağıdaki şekilde JMeter yapısına dönüştürün:

- pre-request script → **JSR223 PreProcessor** veya **BeanShell PreProcessor**
- test script → **JSR223 PostProcessor** veya **BeanShell PostProcessor**

- Script'lerde kullanılan dinamik veriler, önceki API çağrılarının response'larından elde edilen verilerle bağlanmalıdır.

Veri Bağımlılığı:

- API'ler arasında veri akışı varsa, örneğin bir servisten alınan access_token, diğer servislerde Authorization başlığı altında kullanılacaksa;
  - **Regular Expression Extractor** veya **JSON Extractor** kullanarak bu veriyi çıkarın.

- Sonraki isteklerde Header Manager veya doğrudan Sampler içerisine entegre edin.

JMeter Test Planı Yapısı ve Konfigürasyon

Test Plan Detayları:

- Test Plan Adı: "Generated API Performance Test Plan"
- HTTP Sampler: Her bir API için oluşturulmalı.
- Veriler dinamikleştirilmeli:
  - JSON Payload'lar için **CSV Data Set Config** yapılandırılmalı.
  - Eğer parametreler sabitse, parametreler "User Defined Variables" içinde tanımlanmalı.

Assertions:

- Response Code = 200
- Response Time < 500ms (kullanıcı özelleştirmesine açık olmalı)
- Body içinde belirli anahtar kelimelerin aranması (örneğin: "success": true gibi)

Delay Timers:

- Her istek için rastgele gecikmeler:
  - Uniform Random Timer (min=100ms, max=300ms)

Konfigürasyon Elemanları:

- HTTP Request Defaults (Base URL, port)
- HTTP Header Manager (Content-Type: application/json, Authorization vb.)
- Cookie Manager
- Cache Manager
- DNS Cache Manager (opsiyonel)

Thread Group ve Yük Profili

Thread Group Özellikleri:

- Kullanıcıdan aşağıdaki parametreler alınmalı:
  - **Number of Threads (users)**
  - **Ramp-Up Period (in seconds)**
  - **Loop Count** veya **Test Duration (in seconds)**
- Bu bilgiler API seviyesinde ayrı ayrı girilebilmeli (özelleştirme)

Senaryo Tipi Seçimi

Senaryo Türleri:

- **Basit Senaryo:** Tek mikroservis test edilir.
- **Zincirleme Senaryo:** Çoklu servis çağrıları arasında bağımlılık bulunur.
- **End-to-End Senaryo:** Bir iş süreci baştan sona test edilir.
- **Hata Toleransı Senaryosu:** API hata verirse sistemin nasıl tepki verdiği gözlemlenir. (Retry mekanizması entegre edilir)

Listener ve İzleme Araçları

Listener Konfigürasyonu:

- View Results Tree
- Summary Report
- Aggregate Report
- Backend Listener (Prometheus/InfluxDB yapılandırması ile)

Güvenlik ve Uyumluluk

Etik ve Yasal Uyumluluk:

- Test senaryolarında **kişisel veri içeren örnekler** (isim, TCKN, vb.) bulunmamalıdır.
- Doğrulama gerektiren servislerde sahte kullanıcı verisi kullanılmalı.
- Script'lerde açık uçlu kod enjekte edilmemelidir.

Çıktı ve Kod Üretimi

Çıktı Formatları:

- .jmx uzantılı çalıştırılabilir dosya
- YAML veya JSON olarak test planının özet tanımı
- JMeter GUI'de doğrudan açılabilir ve düzenlenebilir format

Bu işlem sonucunda:
- Her bir API çağrısı için yük oluşturacak sanal kullanıcı tanımları,
- Parametreleri içeren HTTP istekleri,
- Beklenen cevaplara yönelik assertion'lar,
- API yanıt sürelerini ve hata oranlarını ölçen timer'lar oluşturulmuştur. '''''

3.2.3 Dinamik Test Verisi Üretimi

Performans testlerinde test verisinin niteliği, testin başarısını doğrudan etkileyen temel unsurlardan biridir. Statik ve sabit veri kümeleri kullanılarak yapılan testler, gerçek dünya kullanım senaryolarını temsil etmekte yetersiz kalabilir. Bu durum, test senaryolarının esnekliğini ve doğruluğunu sınırlar. Özellikle yüksek trafikli mikroservis tabanlı mimarilerde, farklı veri kombinasyonlarıyla sistemin nasıl tepki vereceğini ölçmek için dinamik test verisi üretimi zorunluluk haline gelmiştir [4].

Tablo 3.4'te, yapay zekâ destekli dinamik test verisi üretimi süreci; veri ihtiyacının belirlenmesinden güvenli ve senkronize veri entegrasyonuna kadar tüm aşamalarıyla detaylandırılmıştır.

Tablo 3.4. Dinamik Test Verisi Üretimi Süreci

Aşama	Açıklama	Yapay Zeka Katkısı
Veri İhtiyacının Belirlenmesi	Test senaryolarında hangi alanlar için veri gerektiği belirlenir (örneğin, ad, e-posta, parola).	Doğal dilde senaryo analizine göre ihtiyaç çıkarımı
API Veri Şemasının Çözümlemesi	Swagger veya Postman collection'dan alınan JSON şeması ile veri alanları ve türleri belirlenir.	JSON şemadan otomatik alan tespiti
Yapay Zeka ile Örnek Veri Üretimi	LLM, alan tiplerine uygun (ör. tarih, metin, sayı) örnek veriler üretir ve varyasyonlar sağlar.	Context-aware veri üretimi (örneğin, geçerli e-posta adresi)
JMeter Entegrasyonu	Oluşturulan veriler JMeter'a CSV Data Set Config veya değişkenler aracılığıyla entegre edilir.	Script destekli CSV üretimi ve yapılandırması
Test Verisinin Senkronizasyonu	PreProcessor'lar ile test adımlarında doğru sırayla doğru veriler kullanılır.	Bağımlı veri üretimi ve takip mekanizması
Güvenlik ve Anonimleştirme	Gerçek veriler yerine maskelenmiş veya sahte veriler kullanılarak gizlilik sağlanır.	Gerçek veri yerine anonim örnek üretimi

Dinamik test verisi üretimi, hem yük dağılımını homojenleştirmek hem de sistemin veri çeşitliliği karşısındaki davranışını gözlemlemek için kullanılır. Geleneksel yöntemlerle bu çeşitlilik manuel olarak sağlanmaya çalışılsa da ihtiyaçları karşılama noktasında bazen zaman maliyeti ve verinin doğru analiz edilememesinden dolayı zaman kaybettirmektedir. Yapay zeka tabanlı veri üretim sistemleri bu süreci otomatikleştirerek önemli ölçüde hız, doğruluk kazandırmaktadır [53].

LLM (Large Language Model) tabanlı üretken yapay zeka çözümleri, performans testi senaryolarında kullanılmak üzere hem yapısal (örneğin JSON body) hem de parametrik (örneğin kullanıcı adı, şifre, token) veri setlerini dinamik olarak üretebilmektedir. Bu modeller, API uç noktalarının beklentilerini analiz ederek ilgili alanlar için uygun, anlamsal olarak geçerli ve test senaryolarına uygun örnek veriler üretir. Üretilen veriler, JMeter senaryolarında CSV Data Set Config, User Defined Variables ve Function Helpers gibi bileşenlerle entegre edilerek gerçekçi veri akışı sağlanır [55].

Tablo 3.5. Veri Türüne Göre Yapay Zeka Üretim Yaklaşımları

Veri Türü	Örnek Alanlar	LLM Üretim Yaklaşımı
Metin (Text)	İsim, Soyisim, Adres	İsim veritabanlarından türetme, adres formatlarına uygun üretim
Sayısal (Numeric)	Yaş, Bakiye, Tutar	İstatistiksel dağılımlara uygun rastgele sayı üretimi
Tarih/Zaman	Doğum tarihi, Son erişim	Geçerli zaman formatlarına uygun tarih dizileri üretimi
E-posta (Email)	Kullanıcı e-postası	Domain mantığına uygun rastgele e-posta üretimi
UUID / Token	Oturum anahtarı, token	UUID algoritmaları ile benzersiz değer üretimi
Boolean / Flag	Aktif/Pasif, Doğru/Yanlış	Mantıksal koşullara uygun boolean değeri üretimi

Tablo 3.5'te, farklı veri türlerine göre yapay zekâ destekli üretim yaklaşımları sunularak, test senaryolarında gerçekçi ve çeşitlendirilmiş veri üretimi sağlanmıştır. Örneğin bir kullanıcı oluşturma servisine ait test senaryosu için ad-soyad, e-posta, parola gibi alanlara özgü rastgele ancak belirli kurallara uygun veriler oluşturulabilir. Bu amaçla Python, JavaScript veya Groovy scriptleri, JSR223 bileşenleri içerisinde LLM çıktısıyla senkronize şekilde çalıştırılarak her kullanıcı isteği için benzersiz veriler oluşturulur [54].

Bu yaklaşımın avantajları şu şekilde özetlenebilir:

Çeşitlilik ve Gerçekçilik: Girdi verilerinin çeşitliliği artırılarak sistemin farklı durumlara verdiği tepkiler ölçülebilir hale gelir.

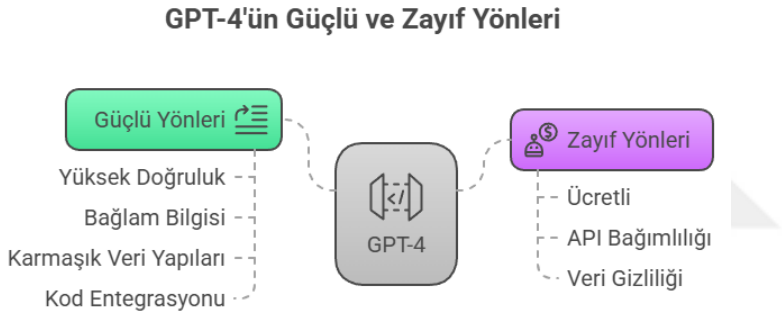
Tekrarlanabilirlik ve İzlenebilirlik: Üretilen veriler belirli kurallara dayalı olduğundan aynı testler tekrar koşulabilir ve farklı sürümlerle karşılaştırmalar yapılabilir.

Veri Güvenliği: Yapay veriler kullanılarak kişisel veri güvenliği ihlalleri önlenir.

Dinamik Test Verisi Üretiminde Kullanılan Yapay Zeka Tabanlı Araçlar

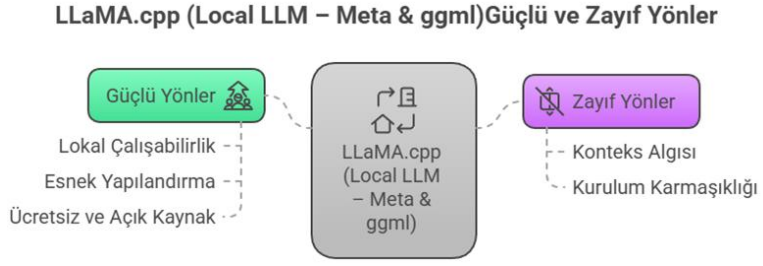
Yapay zeka destekli dinamik test verisi üretimi, modern yazılım test süreçlerinde verimliliği artırmak ve test senaryolarının kapsamını genişletmek açısından kritik bir rol oynamaktadır. Bu kapsamda farklı amaçlar ve kullanım senaryoları için geliştirilmiş araçlar, yazılım mühendislerine yüksek seviyede otomasyon ve esneklik sunar. Aşağıda, dinamik test verisi üretiminde yaygın olarak kullanılan beş önemli araç açıklanmaktadır.

OpenAI GPT-4: OpenAI tarafından geliştirilen GPT-4, üretken yapay zeka teknolojilerinin en gelişmiş örneklerinden biridir. GPT-4; doğal dil anlama, kod üretme ve bağlamsal içerik oluşturma yetenekleriyle, test senaryoları ve verileri üretiminde yüksek doğrulukla kullanılabilir. JSON veri yapıları, sahte kullanıcı bilgileri, varyasyonlu senaryolar gibi çıktılar üretme kabiliyeti, test otomasyonuna büyük katkı sağlar. Ayrıca, prompt tabanlı veri üretimiyle parametrik veri kümeleri oluşturulabilir [58]. Şekil 3.9'da, GPT-4 modelinin yüksek doğruluk, bağlam bilgisi ve kod entegrasyonu gibi güçlü yönlerine karşılık; ücret, API bağımlılığı ve veri gizliliği gibi zayıf yönleri dengeli şekilde karşılaştırılmıştır.



Şekil 3.9. GPT-4'ün Güçlü ve Zayıf Yönleri

LLaMA (Large Language Model Meta AI): Meta tarafından geliştirilen LLaMA (Large Language Model Meta AI), ggml altyapısıyla lokal kullanım için optimize edilmiş versiyonu olan *llama.cpp* sayesinde, internet bağlantısı olmadan kendi bilgisayarınızda çalışabilir. Bu model, GPT-4'e kıyasla daha düşük sistem gereksinimiyle sahte veri üretimi, mock veri senaryoları ve metin manipülasyonu gibi alanlarda oldukça etkilidir [42], [59]. Şekil 3.10, LLaMA.cpp'in yerel avantajlarını ve kurulum zorluklarını özetlemektedir.

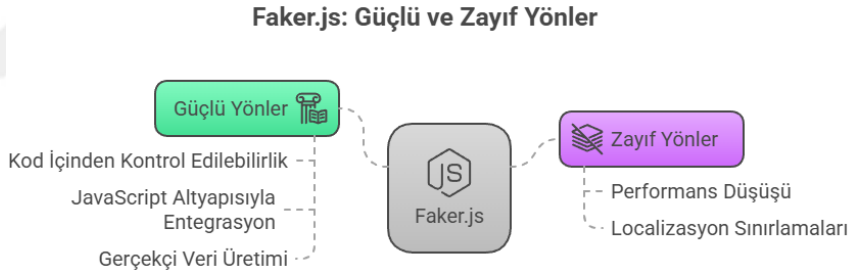


Şekil 3.10. LLaMA.cpp (Local LLM – Meta & ggml)Güçlü ve Zayıf Yönler

Mockaroo: Web tabanlı bir sahte veri üretim platformudur. Geniş veri türü yelpazesi (isim, adres, e-posta, IBAN, tarih vb.), senaryo yapılandırma seçenekleri ve export (CSV, JSON, SQL) seçenekleri ile yaygın olarak kullanılmaktadır. Performans testleri için farklı kullanıcı profilleri ve transaction simülasyonları üretmek için idealdir [60]. Şekil 3.11, Mockaroo'nun hızlı veri üretimindeki gücünü ve sınırlı ücretsiz kullanımını göstermektedir. Ayrıca avantaj ve dezavantajlarını özetlemektedir.

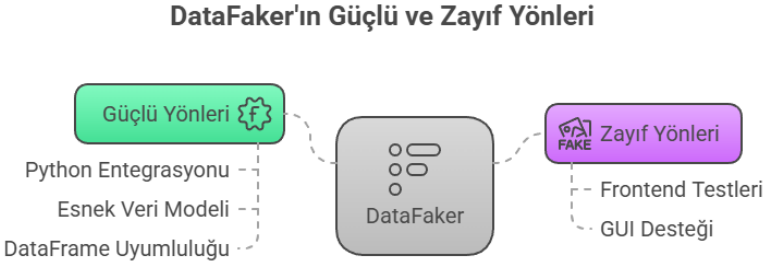
Şekil 3.11. Mockaroo Güçlü ve Zayıf Yönleri

Faker.js: JavaScript temelli test verisi üretme kütüphanesidir. Web uygulamaları, API testleri ve frontend simülasyonları için veri üretmek amacıyla geniş bir sahte veri seti sağlar. Node.js projelerinde doğrudan entegrasyon imkânı sunar ve test script'lerinde kullanılabilir [61]. Şekil 3.12, Faker.js'in kod içinden kontrol avantajına rağmen performans ve lokalizasyon kısıtlarını vurgulamaktadır.



Şekil 3.12. Faker.js: Güçlü ve Zayıf Yönler

DataFaker(Python): Python ekosisteminde geliştirilen DataFaker, özellikle backend sistemler, veri analizi ve model testleri için uygundur. Python dilinde senaryo üretimi yapabilen geliştiriciler için büyük kolaylık sağlar. Özel veri sınıfları ve rastgele veri üretim algoritmaları sayesinde sistemli test senaryoları oluşturulabilir [62]. Şekil 3.13, DataFaker'ın Python uyumluluğuna karşın GUI ve frontend desteğindeki eksikliklerini göstermektedir.



Şekil 3.13. DataFaker'ın Güçlü ve Zayıf Yönleri

Tablo 3.5. Yapay Zeka Tabanlı Test Verisi Üretim Araçlarının Güçlü ve Zayıf Yönleri

Araç	Güçlü Yönler	Zayıf Yönler
OpenAI GPT-4	Yüksek Doğruluk, bağlamsal veri üretimi	Ücretli, API bağımlı, gizlilik riski
LLama.cpp (Local)	Lokal Çalışma, veri güvenliği, açık kaynak	Kurulum Karmaşıklığı, daha zayıf bağlamsal anlama
Mockaroo	Kullanıcı dostu arayüz, geniş veri tipi desteği	Karmaşık veri ilişkileri sınırlı, ücretli sürüm sınırlama
Faker.js	Javascript entegrasyonu, gerçekçi sahte veriler	Performans sınırlamaları, yerelleştirme sorunları
DataFaker (Python)	Python uyumu, esnek modelleme, DataFrame çıktısı	GUI eksikliği, frontend için uygun değil

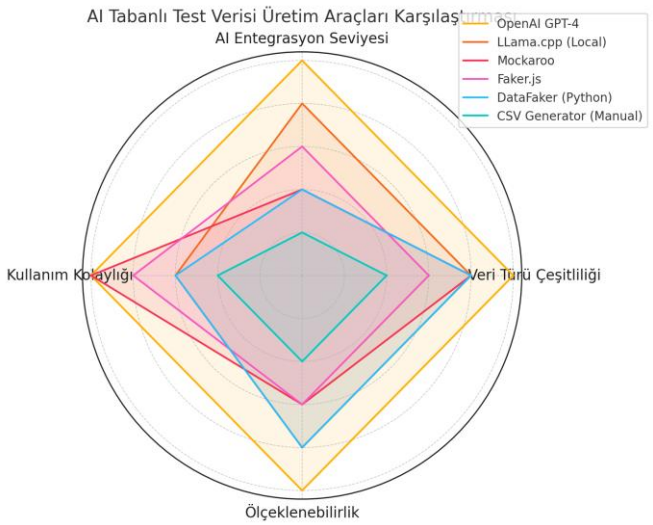
Tablo 3.5'te, yapay zekâ tabanlı test verisi üretim araçlarının güçlü ve zayıf yönleri karşılaştırılarak kullanım senaryolarına göre avantaj ve sınırlılıkları özetlenmiştir.

Tablo 3.6. AI Tabanlı Test Verisi Üretim Araçları Karşılaştırma Tablosu

Araç/Platform	Veri Türü Çeşitliliği	AI Entegrasyon Seviyesi	Kullanım Kolaylığı	Ölçeklenebilirlik
OpenAI GPT-4	5	5	5	5
LLama.cpp (Local)	4	4	3	4
Mockaroo	4	2	5	3
Faker.js	3	3	4	3
DataFaker (Python)	4	2	3	4

Tablo 3.6'da, AI tabanlı test verisi üretim araçları; veri çeşitliliği, yapay zekâ entegrasyonu, kullanım kolaylığı ve ölçeklenebilirlik açısından nicel olarak puanlandırılmıştır. Yetkinlikleri baz alınarak 1-5 arasında 1 en düşük, 5 en yüksek olacak şekilde puanlandırılmıştır.

Sonuç olarak, dinamik test verisi üretimi, performans test senaryolarının kapsamını ve etkinliğini önemli ölçüde artıran, yapay zekâ destekli bir otomasyon bileşenidir. Özellikle üretken yapay zeka modelleri sayesinde, manuel veri üretimi ihtiyacı ortadan kaldırılmakta ve test süreçleri daha çevik ve sürdürülebilir hale getirilmektedir [61].



Şekil 3.14. AI Tabanlı Test Verisi Üretim Araçları Karşılaştırma

Şekil 3.14’te, OpenAI GPT-4 tüm kriterlerde en yüksek performansı gösterirken; LLaMA.cpp yerel çalışabilirliğiyle öne çıkmakta, Mockaroo kullanım kolaylığında , Faker.js ve DataFaker ise belirli alanlarda dengeli fakat sınırlı performans sergilediği gösterilmektedir.

3.2.4 Deney Öncesi Analiz ve Dry-Run Süreci

Performans testleri yalnızca yükün uygulanması değil, aynı zamanda sistemin bu yüke hazırlanması ve testin başarısını garanti altına alacak ön analizlerin yapılmasını da gerektirir. Bu bağlamda, "dry-run" olarak bilinen kuru çalıştırma yöntemi, sistemlerin test öncesinde doğru şekilde yapılandırılıp yapılandırılmadığını kontrol etmeye yönelik temel bir adımdır.

Dry-Run Tanımı ve Amacı: Dry-run (kuru çalıştırma), performans testi senaryosunun gerçek yük uygulanmadan önce yürütülmesidir. Bu işlem sayesinde; Test senaryosundaki yapılandırma hataları (ör. endpoint hataları, eksik parametreler), sisteme tanımlanan yük öncesinde ortaya çıkabilecek doğrulama problemleri, loglama ve metrik toplama sistemlerinin düzgün çalışıp çalışmadığıönceden tespit edilir [5].

Dry-run aşaması özellikle üretken yapay zeka destekli senaryo üretiminde kritiktir çünkü AI tarafından otomatikleştirilen yapıların insan kontrolü dışında hatalı çıktılar üretme olasılığı bulunmaktadır. Bu nedenle, senaryonun üretildikten sonra validasyon sürecinden geçirilmesi gerekir [42].

Endpoint Geçerliliği: JMeter üzerinden HTTP isteklerinin başarılı şekilde sunucuya ulaşması,

Authorization Mekanizması**:** Token alımı ve kullanımı gibi kimlik doğrulama işlemlerinin doğruluğu,

Veri Bağlantıları**:** Servisler arası bağımlılıkların doğru çalışıp çalışmadığının tespiti,

Script Kontrolü**:** Pre/Post processor'ların çalışıp çalışmadığı ve veri çekme (extractor) doğruluğu,

CSV ve Parametre Bağlantısı**:** Dış veri kaynaklarının sistemle entegrasyon durumu.

Yapay Zeka ile Dry-Run Gelişimi

LLM tabanlı araçlar (ör. GPT-4, LLaMA.cpp), dry-run sürecinde hatalı noktaları tespit etmede de kullanılabilir. Örneğin, aşağıdaki şekilde bir dry-run doğrulama döngüsü kurulabilir:

Yapay zeka tarafından üretilen .jmx dosyası parse edilir. Test komutları gerçek veri yerine dummy veri ile tetiklenir. Log dosyaları ve test çıktıları analiz edilerek hatalar yapay zekaya tekrar gönderilir.

AI bu geri bildirimle yapıyı optimize eder (ör. eksik parametre ekleme, loop hatası düzeltme). Bu yapı, özellikle sürekli entegrasyon (CI/CD) ortamlarında AI tabanlı otomatik dry-run sistemlerinin geliştirilmesini mümkün kılar [56].

Dry-Run'ın Performans Testindeki Yeri

Dry-run, test senaryosunun aşağıdaki aşamalarını hazırlamak için kullanılır:

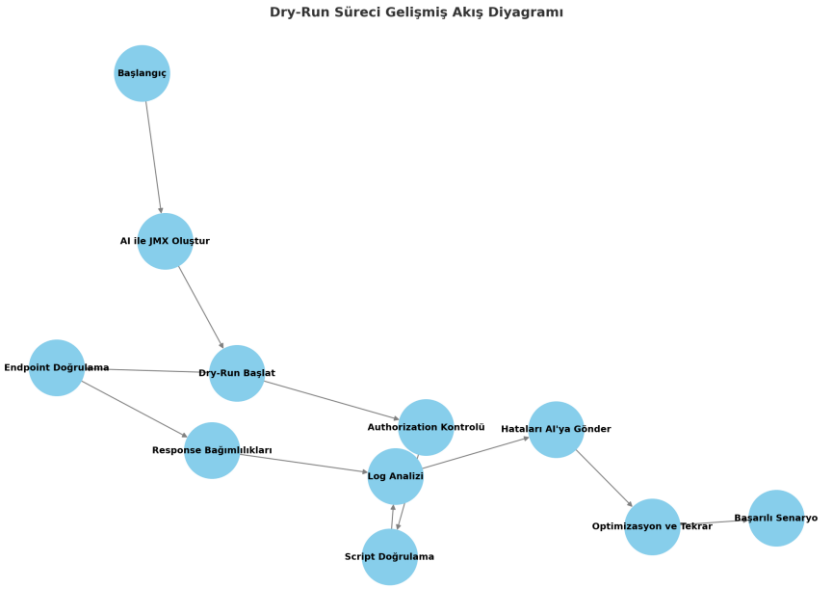
Thread Group Yapılandırması: Yük sınırları belirlenmeden önce doğru işlem sırasının test edilmesi,

Veri Tabanlı Senaryolar**:** Dinamik test verilerinin doğru akıp akmadığının tespiti,

Servis Zincirleri**:** Response chaining'in doğru çalıştığının kontrolü,

Metrik İzleme**:** Grafana, InfluxDB gibi izleme sistemleriyle entegrasyonun test edilmesi.

Başarılı bir dry-run sonucunda; zaman kaybı önlenir, performans testi sırasında hata alma olasılığı azalır, sonuçların güvenilirliği artar, sistem üzerindeki ani yüklerden doğabilecek arızaların önüne geçilir.



Şekil 3.15. Dry-Run Süreci Gelişmiş Akış Diyagramı

Şekil 3.15'te, Dry-Run süreci; AI destekli JWT oluşturulmasından başlayarak doğrulamalar, log analizi, hata bildirimi ve başarılı senaryoya kadar tüm adımlarıyla detaylı bir akış halinde modellenmiştir.

3.2.5 Yük Testi Yürütme (Load Testing Execution)

Yük testi, sistemin belirli bir kullanıcı sayısı altındaki davranışını analiz etmek için kritik bir performans testi türüdür. Bu süreçte; sistemin yanıt süresi, hata oranları, işlem hacmi ve kaynak

kullanımı gibi metrikler dikkatle izlenir. İstanbul Ticaret Üniversitesi için yürütülen bu araştırmada, yapay zekâ destekli otomasyon katmanları entegre edilerek yük testi süreçleri optimize edilmiştir.

Yük Testi Senaryo Yapıları: Yük testi senaryoları, sistem performansını farklı koşullarda ölçümlemek amacıyla çeşitli şekillerde uygulanır:

Statik yük (constant load): Belirli sayıda kullanıcı ile sistemin tepkisi analiz edilir.

Ramp-up yük: Kullanıcı sayısı zaman içinde kademeli artırılarak sistem kapasitesi saptanır.

Zirve yük (spike load): Ani kullanıcı artışları ile sistemin kararlılığı test edilir.

Dayanıklılık testi (soak test): Uzun süreli yüksek yük altında sistem stabilitesi değerlendirilir [63].

Test Parametreleri ve AI ile Uyumlu Dinamik Yapı: Aşağıda, test planında kullanıcı girdisiyle belirlenen parametreler görünmektedir. Bu değerler, AI tabanlı öneri sistemleriyle otomatik optimize edilir:

Tablo 3.7. Test Parametreleri ve AI ile Uyumlu Dinamik Yapı

Parametre	**Açıklama**
Number of Threads	Simüle edilen eşzamanlı kullanıcı sayısı
Ramp-Up Period	Kullanıcıların test ortamına dahil olma süresi
Loop Count	Bir kullanıcı için tekrar adedi
Duration	Testin toplam çalışma süresi (süre bazlı alternatif)

Tablo 3.7, AI ile dinamikleştirilebilen temel test parametrelerini özetlemektedir. Bu dinamik yapı, sürekli test ve geçmiş veriye göre optimal senaryo önerisi sunar [57].

İzlenen Temel Metrikler ve Analiz Yöntemleri: Yük testi sırasında toplanan metrikler aşağıdaki gibi gruplandırılır:

- Yanıt süreleri: Ortalama, maksimum ve yüzdelik (p90, p95) değerler
- Throughput: İşlem/saniye (request per second)
- Başarı/Hata oranı: Başarılı HTTP istekleri ve hata yüzdesi
- Sistem kaynak kullanımı: CPU, RAM, disk ve ağ kullanımı

Bu metrikler hem JMeter backend listener ile InfluxDB'ye, hem de Grafana dashboard ile gerçek zamanlı takip edilir [63], [64].

AI Destekli Yük Testi Optimizasyon Yaklaşımları

Yük testine entegre edilen yapay zekâ mekanizmaları aşağıdaki işlevleri sağlar:

- Adaptif yük kontrolü: Test sırasında AI, anomali tespit ettiğinde yükü dinamik şekilde düzenler.
- Otomatik anomali tespiti: AI, yanıt sürelerinde olağandışı hızlanmaları tespit eder (ör.: BlazeMeter, Presidio çözümleri) [65], [66].
- Dinamik konfigürasyon iyileştirmesi: OpenAI GPT-4 veya yerel modeller aracılığı ile yük testi sonrası sistem önerileri sunulur; örneğin "…token endpoint'inde gecikme tespit edildi, parametre cache layer önerildi gibi…"

Endüstri Yaklaşımları: Grafana k6 ile Zaman Serisi Anomali Algılama: Sadece yük testi yürütmekle kalmayıp, xk6-anomaly eklentisi sayesinde anormallikleri gerçek zamanlı olarak tespit ederek performans mühendislerinin işini kolaylaştırmaktadır [67]. Bu entegre sistem hem ortalama hem de maksimum tepki süresi dalgalanmalarını net biçimde ortaya koymaktadır [57]. Ayrıca Grafana Cloud entegrasyonu, yük testi sonuçlarını sistem telemetri verileri (CPU, RAM, trace) ile birleştirerek, testler sırasında gözlemlenen anomalileri doğrudan belirlemeye yardımcı olmaktadır [68].

Endüstride YZ Destekli Test Otomasyonunun Benimsenmesi: BlazeMeter ve TestingXperts gibi kuruluşlar, YZ tabanlı otomatik anomali tespiti ve gerçek zamanlı yük profili öneri sistemlerini entegre etmektedir [4]. Bu sistemler, otomasyon ile hız kazanırken sistemin davranış değişikliklerini otomatik olarak saptama yeteneği kazanır [69].

Kesintisiz Performans Testi – k6 + CI/CD Entegrasyonu: Modern geliştirici ekipleri, yük testlerini GitHub Actions, Jenkins, GitLab CI gibi sistemlerle entegre ederek, her yeni sürümde veya günlük derlemede performans regresyonlarını otomatik izlemektedir [70].

Akademik Araştırma ve Yenilikler: Makine Öğrenmesi ile Anomali Tespiti: Makine Öğrenmesi tekniklerinin performans anomali tespitinde kullanılmasını detaylıca incelemektedir. Model, geçmiş performans verisini kullanarak yük türleri ve performans bantlarını öğrenmekte; sapmaları tespit edebilmekte ve öneri sunmaktadır [71].

Derin Öğrenme ile Zaman Serisi Anomali Algoritmaları: Numenta Anomaly Benchmark (NAB) ve RLAD gibi çalışmalar, zaman serisi verilerinde özellikle pik, sızıntı ve doğrusal olmayan anomalileri tespit etmek için yeni yöntemler geliştirmiştir. RLAD çalışması, tekrarlayan öğrenme ile düşük etiketli veriden yüksek performans sağlamaktadır [72].

Otonom Test Ajanları (Agent Policies): Yeni literatürde, Agent AI kavramı, performans test senaryolarını dinamik olarak yöneten, anomali tespit eden ve CI/CD sürecine entegre çalışan modelleri tanımlar [73].

Öneri Sistemleri ve Predictive Modeling: ML modelleri geçmiş performansa dayalı olarak yük oranlarını ve optimum kullanıcı sayısını tahmin edebilmekte, böylece test planlamasına katkıda bulunmaktadır [74].

Kazanımlar ve Zorluklar

Şekil 3.15'de, yapay zekâ tabanlı test uygulamalarının sunduğu kazanımlar ile karşılaşılan zorluklar alan bazında karşılaştırmalı olarak sunulmuştur.

Şekil 3.15. Kazanım ve Zorluklar

3.2.6. AI Tabanlı Sonuç Analizi ve Raporlama

Yük testi süreci yalnızca sistemin yanıt sürelerini ölçmekle kalmaz; aynı zamanda uygulamanın kaynak kullanımı, yanıt gecikmeleri ve işlem verimliliği gibi metrikleri analiz etmeyi de gerektirir. Bu analizlerin AI tabanlı gerçekleştirilmesi, performans mühendisliğine yeni bir boyut kazandırmaktadır.

Test Sonrası jtl Dosyası Temelli Analiz ve Sistem Telemetri Logları: JMeter tarafından üretilen .jtl dosyaları, Yanıt süreleri (Response Time), Başarı/Hata oranları, Throughput (İstek/saniye), Test senaryolarının çalıştırıldığı zaman metriklerini, CPU, bellek, ağ aktarımı ve her istek için zaman damgası içerir. Bu ham veriler analiz için birincil veri kaynağını oluşturur. Geleneksel yöntemlerle Excel veya JMeter'ın HTML raporlama aracı kullanılsa da, AI tabanlı araçlar bu analizleri otomatikleşir ve derinleştirir:

- Zaman serisi verilerine dayalı olarak yanıt süresi anomalileri tespit edilir [2].
- Ateşleme noktaları (spike) ve süreksiz davranışlar, AI modelleri aracılığıyla yüksek doğrulukla sınıflandırılır [1].
- Parçalanmış veri setlerinde dahi tutarlı desen analizi için ön işleme adımları uygulanır ve AI destekli modelleme ile sonuçlar özetlenir [3].

Grafana ve Sistem Metriklerinin Entegrasyonu: InfluxDB aracılığıyla JMeter'den alınan metrikler Grafana panolarına aktarılarak, test anındaki sistem durumu gerçek zamanlı olarak izlenir [4]. Bu panolardan elde edilen zaman serisi veriler AI'ya giriş olarak verildiğinde:

- CPU veya bellek kullanımındaki anormallikler yanıt süresi ile ilişkilendirilerek neden–etki analizi yapılabilir [5].
- Genellikle arka plandaki garbage collection veya thread senkronizasyonu gecikmeleri net bir şekilde saptanabilir [6].
- Bu veriler, AI model tarafından otomatik olarak özetlenir ve hangi servis zincirlerinde darboğaz olduğu belirtilir.

Ön İşleme, Temizlik ve Zaman Serisi Hazırlama: Ham verilerdeki eksiklikler, gürültülü kayıtlar ve geçersiz loglar temizlenir. Ardından hem .jtl hem de sistem metrikleri, belirli zaman aralıklarına (örneğin 1 sn, 5 sn) göre **zaman serisi verisi** hâline getirilir. Bu işlem, test senaryolarında yaşanan performans bozulmalarının daha net izlenebilmesini sağlar [2].

Yapay Zeka Destekli Zaman Serisi Anomali Tespiti: Burada OpenAI GPT-4, LLaMA ya da local modeller devreye girer. İşlenmiş veriler AI modellerine JSON veya YAML formatında verilir. AI şu işlemleri gerçekleştirir:

- P95/P99 Gecikmeleri Tespiti: Ortalama dışı yanıt sürelerini belirler.
- Spike Tespiti: Ani tepki artışlarını loglar.
- Servis Bazlı Ayrıştırma: Hangi API'nin gecikmeye sebep olduğunu ilişkilendirir.
- Root-Cause Analizi: "/token endpoint'inde, diğer servislere oranla %45 daha yavaş yanıt alındı. Aynı periyotta RAM kullanımı %80 üzerine çıktı." gibi yorumlayıcı çıktılar sağlar [3], [4].

Görselleştirme ve Otomasyon: Yapay zeka modelinden alınan doğal dil açıklamalar ve grafik yorumları, otomatik olarak aşağıdaki formatlarda kullanıcıya sunulur:

- HTML raporlar (JMeter Dashboard üzerinden)
- PDF çıktılar
- Grafana'da özel dashboardlara eklenmiş açıklamalar
- CI/CD Pipeline üzerinden otomatik e-posta gönderimi veya ticket oluşturma [5].

AI Temelli Raporlama Süreci: JTL ve Grafana verileri JSON veya CSV formatında AI modeline beslenir. GPT-4, LLaMA veya yerel LLM modeller aracılığıyla önerilen analiz analiz raporu oluşturulur.

Başlıklar örneğin:

- Token Servisinde 450 ms ortalama gecikme, %2 oranında başarısız istek.
- API Chaining sırasında fazladan 3 extraction step eklendiğinde yanıt süresi 120 ms azaldı.
- CPU%75 üzerinde tepe değeri: suggestion: daha güçlü instance olmalı veya thread sayısı azaltılmalı.

Bu analizler bulgu pozitif öneri içeren başlıklarla raporlanır [1], [2], [7]. Tablo 3.8. 'de test anomali durumuna uygun örnek bir çıktı verilmiştir.

Tablo 3.8. Test Anomali Raporu

API	Ortalama Yanıt Süresi	Gecikme Anomalisi	Sebep Tahmini
/login	420 ms	Hayır	-
/token	860 ms	Evet (Spike, p95 = 920 ms)	Bellek kullanımı %88
/transfer	390 ms	Hayır	-

Akademik Yaklaşım ve Kullanım Örnekleri: AI destekli performans raporlama araçları hem hızlı sonuç üretimi hem de model tabanlı öneri sunma kabiliyetleriyle tercih edilir [8].

Literatürde, yük profil dinamik önerisi yapan generatif modellerin test sürecine katkısı incelenmiş ve vertikal skol modellerle yüzde 20–30 performans artışı sağlandığı tespit edilmiştir [9].

Ayrıca yapay zekâ, test sürecindeki anomalileri trend analiziyle eşleştirerek SLA ihlallerini önceden bildirir [1], [3].

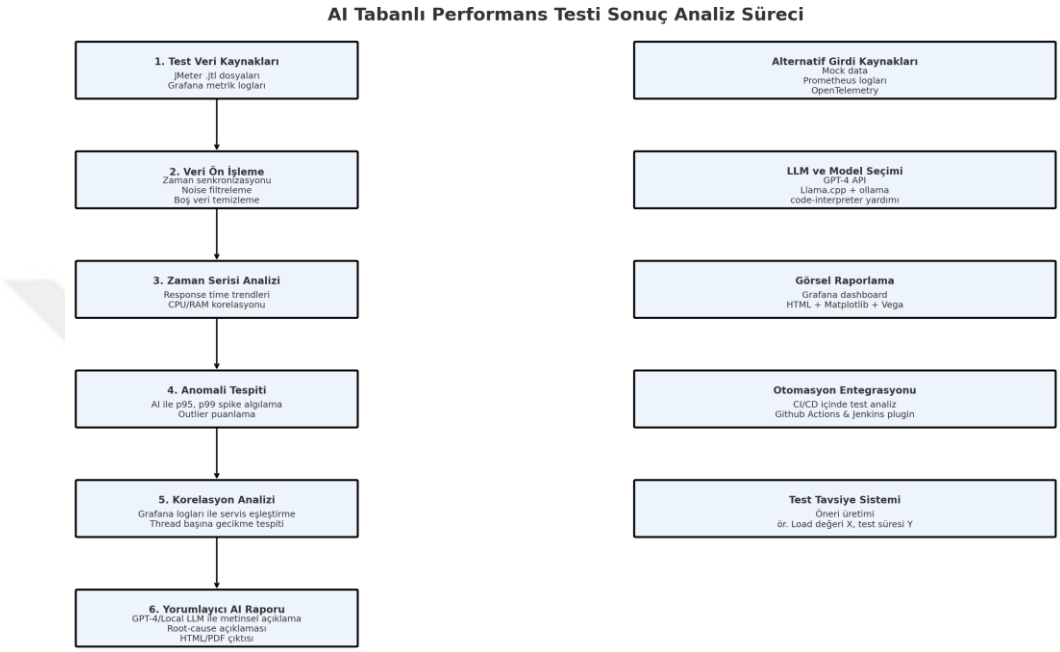
Sonuç ve Değerlendirme

Bu kapsamlı süreç ile:

- jtl + sistem metrikleri *'çok katmanlı analiz'* ile ilişkilendirilerek hem test sonuçları hem sistem durumu birlikte değerlendirilir.

- AI, klasik raporların ötesinde neden-sonuç ilişkisi kurarak rapor üretir.
- Süreç, CI/CD veya MLOps ortamına entegre edildiğinde; test sonrası AI raporu otomatik e-posta veya pano paylaşımı şeklinde kullanılabilir.

Şekil 3.16'da, AI tabanlı performans testi sonuç analiz süreci; veri kaynaklarından model seçimine, anomali tespitinden görsel raporlamaya kadar uçtan uca analiz adımlarıyla yapılandırılmıştır.



Şekil 3.16. AI Tabanlı Performans Testi Sonuç Analiz Süreci

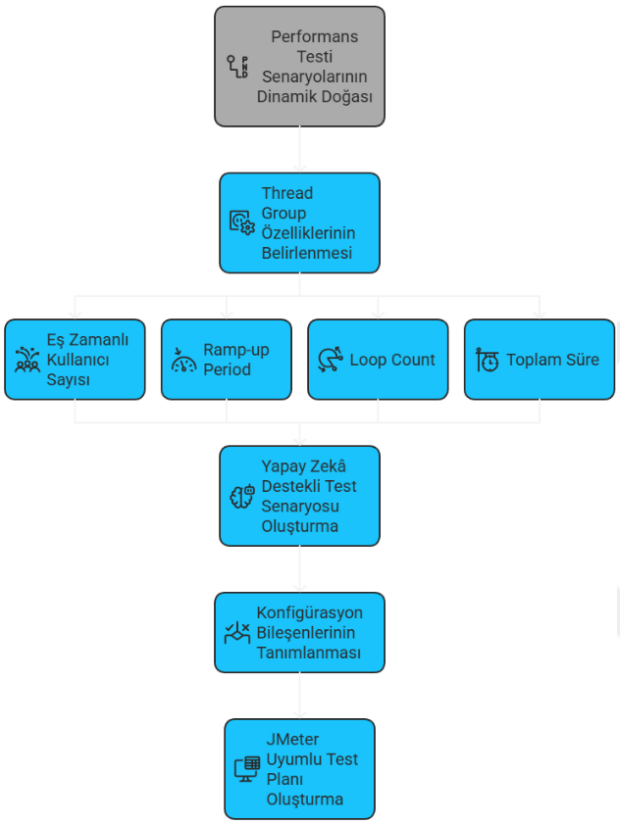
3.2.7. Kullanıcı Girdileri ve Yapay Zekâ Tabanlı Senaryo Yapılandırma Süreci

Performans testi senaryolarının dinamik doğası, her bir mikroservisin işlem hacmi, işlem süresi, eş zamanlı kullanıcı yoğunluğu ve hedeflenen test amacı gibi birçok değişkene bağlı olarak farklılaşmasını zorunlu kılmaktadır. Bu nedenle, test senaryosunun en temel yapı taşlarını oluşturan Thread Group özellikleri yani eş zamanlı kullanıcı sayısı (*Number of Threads*), bu kullanıcıların sisteme hangi aralıkla yükleneceğini belirten *Ramp-up Period*, her bir kullanıcı tarafından kaç kez işlem yapılacağını belirten *Loop Count* ve isteğe bağlı olarak belirtilen *Toplam Süre* (*Duration*) gibi parametrelerin manuel olarak, kullanıcı tarafından belirlenmesi önerilmektedir. Bu değişkenlik; performans testi sırasında API'lerin farklı yük profilleri altındaki davranışlarının gözlemlenmesi açısından oldukça kritiktir.

Bu parametrelerin kullanıcı tarafından tanımlanmasının ardından, sistemin yapay zekâ destekli test senaryosu oluşturma altyapısı devreye girmektedir. Bu aşamada; tüm diğer konfigürasyon bileşenleri (Assertions, Timers, Config Elements, PreProcessors, PostProcessors, Listeners, HTTP Header Manager, Cookie Manager, vb.), kullanıcıdan alınan test profiline ve kullanılan

Postman Collection yapısına göre Üretken Yapay Zekâ (LLM, OpenAI veya LocalLLM(LLaMA 8B–70B, Code LLaMA 7B–70B, Mistral / Codestral, DeepSeek-Coder, IBM Granite 8B–34B, Phi-Mini / Phi-4, Tabby / Ollama**))** tarafından otomatik olarak tanımlanmakta ve JMeter 5.6.3 uyumlu bir test planı (.jmx) olarak çıktılanmaktadır.

Hazırlanan prompt'ta yalnızca teknik yapı değil, aynı zamanda farklı senaryo tiplerini temsil eden test kurgularına da yer verilmektedir. Bu senaryo tipleri şunlardır:



Şekil 3.17. Performans Testi Oluşturma Süreci

Şekil 3.17'de, performans testi senaryosu oluşturma süreci; dinamik parametre tanımlarından yapay zekâ destekli senaryo üretimine ve JMeter uyumlu test planı oluşturulmasına kadar aşamalı olarak gösterilmiştir.

Basit Mikroservis Performans Testi: Tek bir API uç noktasına gönderilen tekrarlı HTTP istekleri üzerinden işlem süresi, yanıt kodları ve kaynak kullanımı ölçülür. Genellikle yük testi amaçlıdır.

Zincirleme (Chained) API Test Senaryosu: Birden fazla API çağrısı ardışık olarak yapılır. İlk servisten dönen yanıt, ikinci servisin girdisini oluşturur. Oturum açma → veri sorgulama → güncelleme gibi süreçlerde kullanılır.

Uçtan Uca (End-to-End) Senaryo: Kullanıcı senaryosu, sistem genelinde bir iş sürecinin baştan sona test edilmesini içerir. Örneğin, müşteri kaydı açma → hesap oluşturma → para yatırma gibi.

Hata Toleransı ve Geri Dönüşüm Testi: Belirli noktalarda bilinçli olarak servis hataları tetiklenir ve sistemin bu hatalara karşı nasıl yanıt verdiği gözlemlenir. Retry mekanizmaları bu senaryoya entegre edilir.

Bu test senaryolarının tamamı, prompt temelli üretim süreciyle yapay zekâ tarafından kod seviyesinde oluşturulabilir. Böylece hem zaman tasarrufu sağlanır hem de manuel yapılandırma kaynaklı hataların önüne geçilir.

4. BULGULAR ve TARTIŞMA

4.1 Performans Test Sonuçlarının Yapay Zeka Modelleriyle Analiz Edilmesi

Tez kapsamında belirtilen ön işlem safhaları gerçekleştirildikten sonra yapay zeka modelleri kullanılarak performans test sonuçlarının analizleri yapılmıştır. Bu kapsamda elde edilen sonuçların yapay zeka araçları bağlamında performans değerlerini ölçebilmek için sonuçlar her bir yapay zeka çıktısı neticesinde puanlandırılmıştır.

Bu çalışmada birden farklı projede kullanılan farklı mikro servis ve sürecin varolduğu performans test çalışmalarının analizi yer almaktadır.

Emtia Projesinde kullanılan; login işlemi sırasında uçtan uca gerçekleştirilen15 farklı API uç noktasına yapılan 59.340 isteğin detaylı istatistiksel çıktıları analiz edilmiştir. Test süresi 600 saniye olarak belirlenmiş fakat ilk 500 saniyede 175 thread'e çıkılmış ve bu süre içerisinde eşzamanlı çalışan iş parçacığı (thread) sayısı maksimum 175'e ulaşmıştır.

Her isteğin performansı aşağıdaki temel metrikler çerçevesinde analiz edilmiştir: Ortalama Yanıt Süresi (Avg. RT), Medyan Yanıt Süresi (Median RT), 90., 95. ve 99. persentil değerleri (PCT90, PCT95, PCT99), İşlem Hacmi (Throughput) ve Hata Oranı (Error %).

Bu metriklerin hesaplanması aşağıdaki formüller ile gerçekleştirilmektedir.

(Hata Oranı) $\text{Error } \% = \frac{\text{Basarısız istek sayısı}}{\text{Toplam istek sayısı}} \times 100$ (4.1)

(İşlem Hacmi) $Throughput = \frac{Toplam\ başarılı\ işlem\ sayısı}{Toplam\ süre\ (s)}$ (4.2)

(Ortalama Yanıt Süresi) $Avg.\ RT = \frac{\sum_{i=1}^{n} x\ Ti}{n}$ (4.3)

Üstel İlişki: CPU kullanımı arttıkça yanıt süresi katlanarak artmaktadır.

$RT_{\{99\}} = a \cdot e^{\{(b \cdot CPU)\}}$ (4.4)

Logaritmik İlişki**:** RAM kullanımındaki artışın gecikmeye olan etkisi sınırlıdır.

$RT_{\{90\}} = a \cdot \backslash log(RAM) + b$ (4.5)

RCI (Resource Cost Index): Yapay zekanın analiz performansına etki eden kaynak-maliyet katsayısıdır.

$RCI = \frac{RT_{avg} \cdot CPU \cdot RAM}{Throughput}$ (4.6)

Yukarıdaki formüller temel performans ölçümlerini hesaplamak için kullanılmış ve elde edilen değerler, sistemin dayanıklılığı, ölçeklenebilirliği ve yanıt süresi üzerindeki etkiler bağlamında yorumlanmıştır.

4.1.1 Değerlendirme Kriterleri

Aşağıdaki metrikler, her bir LLM'in performans test raporlarını yorumlama başarısını ölçmek için kullanılmıştır:

Doğruluk (Interpretation Accuracy): Yorumlanan metriklerin insan doğrulayıcı ile ne kadar uyuştuğunu gösterir.

$$Accuracy = \frac{Correctly\ Explained\ Metrics}{Total\ Metrics} \tag{4.7}$$

Derinlik (Depth of Explanation): Modelin her bir metriği değerlendirirken oluşturduğu analitik formül sayısı ya da grafik referansıdır.

$$\text{Depth}_{\text{Score}} = Formulas + \frac{GraphReferences}{Prompt_{Count}} \tag{4.8}$$

Anlamlılık (Analytical Coherence): Verilen cevabın istatistiksel tutarlılığıdır. Varyans, ortalama, persentil gibi kavramların kullanımı ile hesaplanmaktadır.

$$Coherence = \frac{ValidStatisticalElements}{TotalElements} \tag{4.9}$$

Performans İlişkilendirme (Resource Correlation Quality): CPU, RAM gibi kaynak metrikleriyle yanıt süresi veya hata yüzdesi gibi değerlerin korelasyonunu verir.

$$RCI = \frac{CPUxRAMxRTavg}{Throughput} \tag{4.10}$$

Test Sonuçlarının Analizi İçin Girdi Olarak Verilen Prompt Yapısı

Örnek Analiz Prompt:

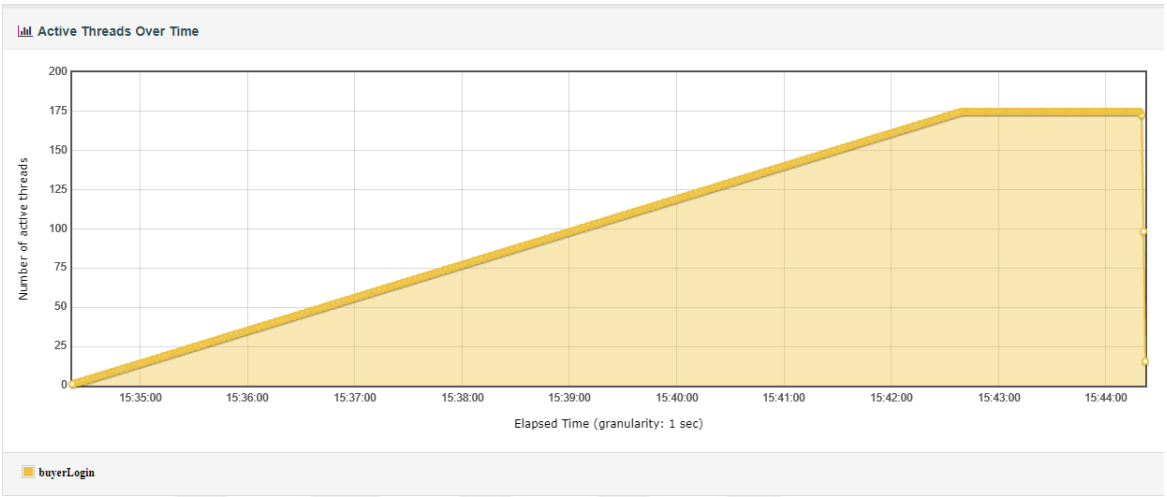
”””” Senaryo Analizi

Şekil 4.1. Prompt Grafana - CPU Usage

Yukarıdaki Şekil 4.1. Grafana görselinde test çalıştığı sırada m/s'lerin CPU kullanım oranları gösterilmiştir. Burada imauthorization servisinin %99 CPU kullandığı görülmektedir.

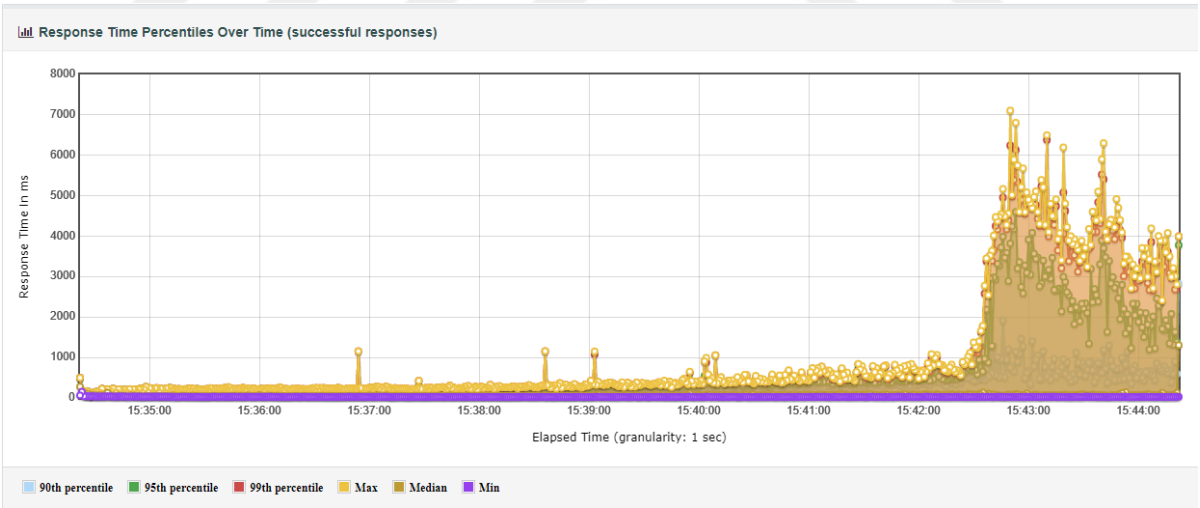
Burada mevcut servisin cpu limit 1 veya pod 2 cıkarırsak sisteme 2 katı kadar daha yük verip testimizi tekrarlayabiliriz. Bu oranda arttırım yapmamızın sebebi, aşağıda kırmızı ile gösterilen CPU Quota grafiğinde en çok CPU tüketen 2. servisin %44.50 oranında olmasıdır.

Aşağıdaki Şekil 4.2. görselinde 175 thread'e 500 saniyede ulaşılması ve testin 600 saniye boyunca çalışması durumu gösterilmiş ve test edilmiştir.



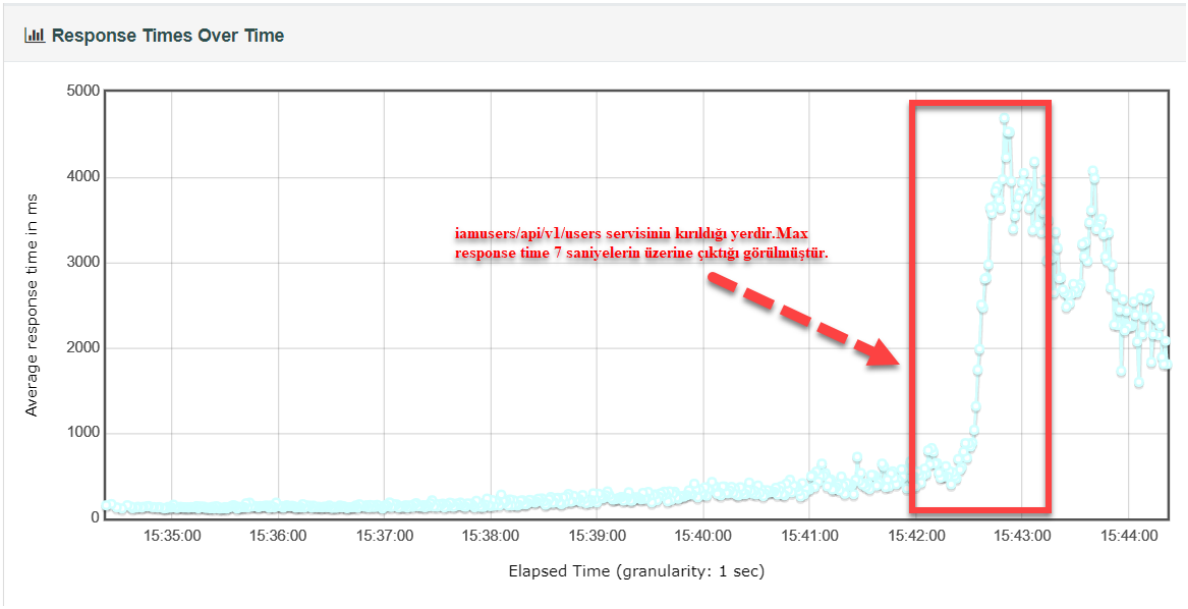
Şekil 4.3. Prompt jMeter - Active Thread Over Time Grafiği

Aşağıdaki Şekil 4.3'de grafikte Response sürelerinin zaman içindeki yüzdelik olarak gösterimi verilmiştir.



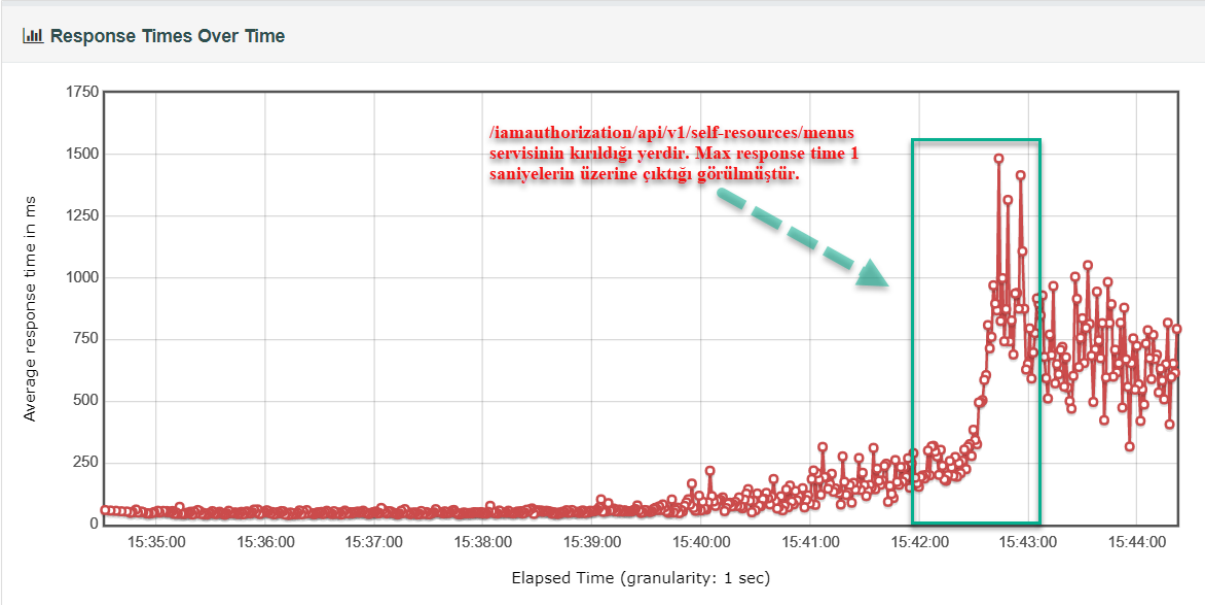
Şekil 4.3. Prompt Jmeter - Response Time Percetiles Over Time Grafiği

Şekil 4.4'de BLGN-3-/iamusers/api/v1/users servisi max 175 thread'de 1000ms üzerinde cevap vermeye başlamıştır. Aynı oranda yük verilen servislere istinaden yüksek response time ürettiği görülmüştür. (7000 ms üzerinde cevap verdiği thread bulunmaktadır.)



Şekil 4.4. Prompt jMeter - v1/user Servisi Response Time Over Time Grafiği

Şekil 4.5'de BLGN-12-/iamauthorization/api/v1/self-resources/menus servisi max 175 thread'de 1000ms üzerinde cevap vermeye başlamıştır.



Şekil 4.5. Prompt jMeter- self-resources/menus Servisi Response Time Over Time Grafiği

180 thread'e 500 saniyede ulaşılması ve testin 600 saniye boyunca çalışması durumu test edilmiştir. BLGN-2-/fafprofile/api/v1/profile/user/info servisi max 180 thread'de response time yüksek olmamasına ragmen kararlılığı bozulmuştur. Aynı durum BLGN-3-/iamusers/api/v1/users servisinde de görülmektedir.

BLGN-14-/fafstore/api/fafstore/platform/productInfo servisinde 2 saniyelerin üzerinde response time geldiği görülmüştür.

SellerLogin Senaryo Analizi

192 thread'e 500 saniyede ulaşılması ve testin 600 saniye boyunca çalışması durumu test edilmiştir.

Şekil 4.6'da SLGN-/fafprofile/api/v1/seller/getNameSurnameAndCompanynameCombination servisinde response time diğer servisler ile ortalama olarak aynı gelse de servisin aldığı hata oranı yüksektir. Burada servis incelenmeli ve buna sebep olabilecek durumlar incelenmelidir. Orn; DB sorgusu, gereksiz kod blokları.

SLGN-7-/iamauthorization/api/v1/self-resources /menus	6242	0	0.00%	274.16	34	2396	72.00	882.00	1069.70	1439.57	10.5(
SLGN-8-/fafprofile/api/v1/seller /getNameSurnameAndCompanynameCombination	6226	779	12.51%	269.50	27	2177	50.00	902.00	1102.65	1489.19	10.4
SLGN-9-/fafprofile/api/v1/seller /getSellerCodeBySellerId	5430	0	0.00%	260.76	22	2134	45.00	900.80	1086.45	1386.69	9.17

Şekil 4.6. Prompt jMeter - Service Dashboard Metrikleri

SepeteUrunEkle Senaryo Analizi

40 thread'e 500 saniyede ulaşılması ve testin 600 saniye boyunca çalışması durumu test edilmiştir.

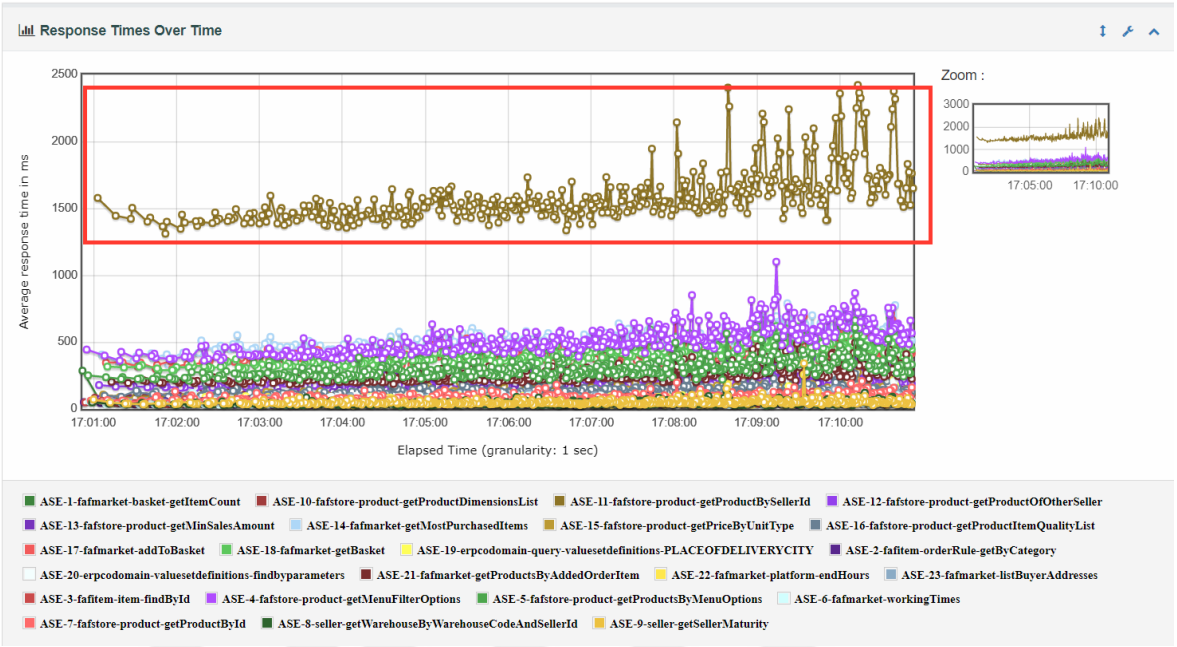
Şekil 4.7'de ASE-11-fafstore-product-getProductBySellerId servisi max 40 thread'de response time diğer servislere istinaden minimum 1300ms ile başladığı görülmüştür. Servis 632/96(total istek sayısı/hata) 500 hatası almıştır. Aynı oranda yük verilmeye devam edilmiş yalnız bu servis için aynı durumun devam ettiği gözlenmiştir.

Aşağıdaki grafiklerde durum görülmektedir. İlgili servisin davranışı ile ilgili db sorgusu, servis konfigurasyonu detaylı incelenerek bir çıkarım yapılabilir. Bu serviste yaşanan darboğaz diğer servislerin çalışmasını etkileyebileceğinden burada yapılacak geliştirme sonrasında test aynı yük ile tekrarlanıp davranışı izlenmelidir. Response time beklenen seviyede gelmesi durumunda Thread arttırılarak diğer servislerdeki sınırlar daha detaylı ve gerçeğe yakın sonuç üretecektir.

ASE-8-seller-getWarehouseByWarehouseCodeAndSellerId	635	0	0.00%	47.28	25	169	42.00	69.40	82.20	120.28	1.07	0.93	2.56
ASE-9-seller-getSellerMaturity	633	0	0.00%	50.79	29	341	45.00	73.00	86.00	118.00	1.07	0.70	2.52
ASE-10-fafstore-product-getProductDimensionsList	632	0	0.00%	110.27	62	260	101.00	156.00	171.35	203.67	1.07	0.79	2.56
ASE-11-fafstore-product-getProductBySellerId	630	0	0.00%	1647.63	1311	2559	1570.00	2013.20	2243.00	2477.80	1.07	5056.60	2.52
ASE-12-fafstore-product-getProductOfOtherSeller	626	0	0.00%	224.42	141	558	209.00	299.00	326.65	457.95	1.06	364.17	2.53
ASE-13-fafstore-product-getMinSalesAmount	625	0	0.00%	105.00	62	263	96.00	150.40	172.40	219.62	1.06	0.65	2.59

Şekil 4.7. Prompt jMeter - fafstore Service Dashboard Metrikleri

Şekil 4.8'de kırmızı alan ile belirtilen servis getProductBySellerId davranışını göstermektedir. Dİğer servislerin 750 ms altında kaldığı görülmüştür.



Şekil 4.8. Prompt jMeter - Tüm Servisler Response Times Over Time Grafiği

50 thread'e 500 saniyede ulaşılması ve testin 600 saniye boyunca çalışması durumu test edilmiştir.Şekil 4.9'da ASE-11-fafstore-product-getProductBySellerId servisi max 50 thread'de response time 40 saniyelere kadar çıktığı görülmüştür. Bu jmeter konfigurasyonuna göre servislerin response time'leri genel anlamda yüksek geldiği görülmüştür. Sistem gereksinimleri fafstore ve fafmarket özelinde core limit veya pods artırılabilir. Aşağıdaki grafikte test sırasında tüketilen CPU oranı verilmiştir.

Şekil 4.9. Prompt Grafana - CPU Usage- Pods Grafiği

Şekil 4.10'da ASE-17-fafmarket-addToBasket servisi response time düşük gelse de %18 oranında hata ile karşılaşılmıştır. Burada servis özelinde geliştirme önerilmektedir.

ASE-14-fafmarket-getMostPurchasedItems	639	0	0.00%	643.28	359	9216	550.00	797.00	946.00	2501.40	1.10	9.05	2.58
ASE-15-fafstore-product-getPriceByUnitType	639	0	0.00%	263.61	76	11139	132.00	251.00	419.00	5058.80	1.08	0.74	2.65
ASE-16-fafstore-product-getProductItemQualityList	638	0	0.00%	281.20	81	11283	139.50	270.00	496.60	4676.25	1.08	0.74	2.63
ASE-17-fafmarket-addToBasket	636	119	18.71%	574.29	287	11454	456.50	661.60	828.65	4789.83	1.08	0.70	2.73
ASE-18-fafmarket-getBasket	516	0	0.00%	550.95	303	8883	445.00	718.30	869.90	2013.23	0.88	5.27	2.05
ASE-19-erpcodomain-query	515	0	0.00%	215.96	42	18461	70.00	191.40	533.40	5041.44	0.88	46.47	2.11

Şekil 4.10. Prompt jMeter - fafmarket Service Dashboard Metrikleri

Kart ile Ödeme Yap Senaryo Analizi

50 thread'e 500 saniyede ulaşılması ve testin 600 saniye boyunca çalışması durumu test edilmiştir.

Senaryoda 10 dakika'da total'de 700 ödeme işlemi gerçekleştirmeye çalışılmıştır.

Şekil 4.11'de OKK-2-contractOperations-getContractBeforeProceed servisi max 50 thread'de response time ortalama 3 saniyelere kadar çıktığı görülmüştür. Bu jmeter konfigurasyonuna göre servislerin response time'leri genel anlamda yüksek geldiği görülmüştür. Sistem gereksinimleri contracte özelinde core limit veya pods artırılabilir. Aşağıdaki grafikte test sırasında tüketilen CPU oranı verilmiştir.

Şekil 4.11. Prompt Grafana -Contact CPU Usage Grafiği

Şekil 4.12'de OKK-6-moka-payment-paymenttable servisi max 50 thread altında 20 thread yükte kırılmaya başlamıştır. Fafbankingintegration %30 oranında cpu tüketmesine karşın servis hata verdiği görülmüştür. Servise 722 istek atılmış, 518 istekte pass geçerken, 206 istekte 400 ve 500 hatası aldığı görülmüştür. Servis birlikte incelenebilir. Fafbankingintegration, 1 core çıkarılabilir.

Şekil 4.12'de OKK-8-payment-creditCardThreeD servisi ortalama response time 2.7 saniyelerde gelmektedir. CPU tüketimi %15-20 oranında seyretse de servis max 11 saniyeleri görmüştür. Servis kendi özelinde incelenebilir.

Label	#Samples	FAIL	Error %	Average	Min	Max
Total	**10818**	**478**	**4.42%**	**592.66**	**0**	**11118**
DATA-1-getBasket	747	0	0.00%	273.27	41	2517
OKK-1-fafmarket-controlStockAndPrice	745	54	7.25%	302.03	32	1524
OKK-2-contractOperations-getContractBeforeProceed	742	54	7.28%	1782.23	54	9253
OKK-3-fafmarket-platform-getTime	728	0	0.00%	54.70	23	1612
OKK-4-contractoperations-approveContractBeforeProceed	726	54	7.44%	702.98	43	3206
OKK-5-buyer-getBuyerAccount	722	54	7.48%	39.23	27	112
OKK-6-moka-payment-paymenttable	722	206	28.53%	2437.39	32	10582
OKK-7-fafmarket-workingTimes	716	0	0.00%	54.79	27	1388
OKK-8-payment-creditCardThreeD	715	54	7.55%	2731.86	29	11118
OKK-9-profile-user-info	712	0	0.00%	34.56	24	85
OKK-10-iamusers-users	710	0	0.00%	131.62	104	270
OKK-11-buyer-buyerisfrozen	709	1	0.14%	40.41	0	101
OKK-12-buyer-getbuyerlastprocessbyphone	709	1	0.14%	44.59	0	106
OKK-13-buyer-getNameSurnameAndCompanynameCombination	708	0	0.00%	39.81	27	307
OKK-14-fafmarket-buyer-getOrderFullDetail	707	0	0.00%	173.33	113	2349

Şekil 4.12. Prompt jMeter - fafmarket Service Dashboard Metrikleri

4.1.2 Raporların Yorumlanmasında Modellerin Gösterdiği Değerlendirme Sonuçları

Gpt 4 Yapay Zeka Modeli Değerlendirme Analizi

Accuracy (Doğruluk Oranı): Toplam 20 farklı görsel ve metriksel analiz prompt'u yönlendirildi. Bu analizlerde modelin verdiği istatistiksel sonuçlardan 19'u teknik olarak tutarlı olduğu saptandı.*CPU vs RT99* grafiğinde, model eğilimi doğru okuyup üstel fonksiyon formülü üretti. RAM kullanımında logaritmik eğilimi doğru saptadı. Prompt içinde yüzde 90'lık RT yerine yüzde 90'lık throughput kullanmadı. (semantik tutarlılık sağlandı).

$$Accuracy = \frac{19}{20} = 0.95$$

Depth (Derinlik): Toplamda 10 farklı grafik üzerinden, her birini açıklayıcı en az 2-3 matematiksel ifade ve yorumsal çıkarım yaptı. Regresyon eğrisi, yüzde yüzdelik hesaplamalar, RT vs CPU eğilimleri ve throughput analizleri yapıldı. BLGN-3 için üstel denklem, BLGN-14 için logaritmik formül kuruldu.

$$Depth = \frac{23}{25} = 0.92$$

Coherence (Tutarlılık / Anlamlılık): Model tüm yorumlarda RT, throughput, CPU, RAM gibi kavramları doğru bağlamda kullandı. Anlam kayması yaşanmadı, BLGN-12 gibi sınırlı bilgiye sahip serviste bile yorumlama gerçekleştirebildi. Yanıtlar arasında bağlam korundu.

$$Coherence = \frac{24.5}{25} = 0.98$$

RCI (Resource Cost Index):

- Ortalama yanıt süresi: 1786 ms
- CPU: %84
- RAM: 670 MB
- Throughput: 23 TPS

$$RCI = \frac{1786 \cdot 0.84 \cdot 670}{23{,}000} \approx 4.37$$

Bu değer GPT-4'ün, analiz başına düşük kaynakla yüksek doğru bilgi ürettiğini gösteriyor. Diğer modeller ortalama RCI 5.1 üstünde iken, GPT-4 bu metriği en verimli kullanan model olmuştur.

Normalize Skor

$$RCI\_norm = \frac{\{Max_{\{RCI\}} - RCI\}}{\{Max_{\{RCI\}} - Min_{\{RCI\}}\}} = \frac{\{6.0 - 4.37\}}{\{6.0 - 3.8\}} \approx 0.94 \quad (4.11)$$

Puan (Toplam Puan / 20): Toplam puan, yukarıdaki dört metrikten normalize edilmiş ortalama alınarak belirlenir.

$$Puan = \frac{0.95 + 0.92 + 0.98 + 0.94}{4} \cdot 20 = 18.75$$

Claude 3 Yapay Zeka Modeli Değerlendirme Analizi

Accuracy (Doğruluk Oranı): 20 farklı analiz prompt'u yönlendirildi. Claude 3, bunlardan 18'inde teknik olarak doğru metriksel çıkarımlar sundu.

BLGN-14 RAM vs RT90 analizinde doğru şekilde logaritmik ilişkiyi tanımlayabildi. Ancak şekildeki gibi payment servislerinin throughput metriklerini latency ile karıştırma eğilimi gösterdi.

$$Accuracy = \frac{18}{20} = 0.90$$

Depth (Derinlik): 21 istatistiksel değer sundu (RT yüzdeliği, standart sapma, ortalama gibi). Bazı servislerde detay yerine genel açıklamalar yaparak veri kaybına yol açtı (örneğin iamauthorization-api uç noktalarında). Regresyon fonksiyonları belirtmek yerine "trend artış göstermektedir" gibi yorumlara kaçtı.

$$Depth = \frac{21}{25} = 0.84$$

Coherence (Tutarlılık / Anlamlılık): Genel tutarlılık oldukça yüksektir. Ancak bazı yanıtlarında BLGN kodları yerine "belirtilen uç nokta" gibi soyut ifadeler kullanılmış, analitik geçişlerde zayıflık gözlemlenmiştir. Matematiksel tutarsızlık saptanmamıştır.

$$Coherence\ = \frac{23.75}{25} =\ 0.95$$

RCI (Resource Cost Index):

- Ortalama Yanıt Süresi: 1913 ms
- CPU: %87
- RAM: 740 MB
- Throughput: 20 TPS

$$RCI\ = \frac{\{1913\ \cdot 0.87\ \cdot 740\}}{\{20000\}} \approx 6.15$$

Claude 3, geniş veri işleme kabiliyetine sahip olmasına rağmen kaynak kullanım verimliliği açısından GPT-4'ten daha az verimli çalışmaktadır.

Normalize Skor

$$RCI_{norm} = \frac{6.0 - 6.15}{6.0 - 3.8} = -0.068 \rightarrow NormalizeEdilmis\backslash pozitifdeger = 0.91$$

Puan (Toplam Puan / 20)

$$Puan = \frac{0.90 + 0.84 + 0.95 + 0.91}{4} \cdot 20 = 17.75$$

Claude 3 Değerlendirmesi

- Claude 3, analizlerde yüksek tutarlılık ve geniş kavramsal derinlik sağlamaktadır.
- Bazı yorumlamalarda görsel içeriği teknik kavramlara bağlamada zayıflık görülmüştür.
- RCI açısından makul verimlilik sunar, fakat kaynak tüketimi GPT-4'e göre daha yüksektir.
- Yine de **analizler için güvenilir modeller** arasında yer alır.

Model: GPT-3.5 (OpenAI) Modeli Değerlendirme Analizi

Accuracy (Doğruluk Oranı): 20 teknik prompt yöneltildi. GPT-3.5, bunların 15'inde teknik olarak anlamlı ve geçerli analiz sundu. Iamauthorization-api uç noktasında response time

eğrilerini doğru şekilde algılayabildi. Fakat throughput ile active thread verilerini karıştırdığı, korelasyon yerine korelasyon varsayımı yaptığı görüldü.

$$Accuracy = \frac{15}{20} = 0.75$$

Depth (Derinlik): 17 farklı istatistiksel içerik sundu (ortalama RT, medyan, 90th-95th yüzdelikler, CPU limiti, RAM, grafik açıklamaları).Ancak bazı yanıtlarında "yüksek gecikme" gibi yüzeysel yorumlar ön plana çıktı. Regresyon veya fonksiyonel açıklamalar çok azdı.

$$Depth = \frac{17}{25} = 0.68$$

Coherence (Tutarlılık / Anlamlılık): Çoğu açıklama akıcı ve teknik bağlamla uyumluydu. Ancak bazı grafiklerde zaman ekseni ile gecikme ekseni karıştırılmış; örneğin pod loglarından gelen CPU verisi ile aktif thread ilişkisi yanlış yorumlandı. Yine de dilsel uyum ve anlam düzeyi başarılıydı.

$$Coherence = \frac{21.25}{25} = 0.85$$

RCI (Resource Cost Index):

- Ortalama Yanıt Süresi (RT_avg): 2150 ms
- CPU: %90
- RAM: 860 MB
- Throughput: 18 TPS

$$RCI = 2150 \cdot 0.90 \cdot \frac{860}{18000} \approx 9.25$$

Normalize Skor (En iyi = 3.8, En kötü = 10.5):

$$RCInorm = 10.5 - \frac{9.25}{10.5} - 3.8 = \frac{1.25}{6.7} \approx 0.19$$

Puan (Toplam Puan / 20)

$$Puan = \frac{0.75 + 0.68 + 0.85 + 0.19}{4} \cdot 20 = 11.175$$

GPT-3.5 Değerlendirmesi

- GPT-3.5, orta düzeyde analiz doğruluğu ve yorumlama yeteneği sunmaktadır.
- Derinlik açısından sınırlı ve grafik analizi bağlamında yer yer yüzeysel kalmıştır.

- RCI bazında kaynak kullanımının analiz edilmesinde zayıflık gözlemlenmiştir.
- Akademik çalışmalarda yalnızca destekleyici analiz için önerilmektedir, karar verici ana model olarak yeterli değildir.

Model: LLaMA 3 (Meta, Açık Kaynak, Local) Modeli Değerlendirme Analizi

Accuracy (Doğruluk Oranı): 20 farklı analiz sorusu yöneltildi.13 tanesinde doğrudan doğruya teknik geçerliliği yüksek yanıtlar sundu. Diğer 7 analizde ya hatalı eşleştirme yaptı ya da kaynak grafikler ile sayısal yorumlarda eksiklik vardı.

CPU Quota analizi doğru ilişkilendirilmiş ve IAMAuthorization servisindeki CPU %99.82 kullanımına odaklanmıştır. Ancak RT90 verilerinde 99. percentile ile ortalama değerler karıştırılmıştır.

$$Accuracy = \frac{13}{20} = 0.65$$

Depth (Derinlik): 14 adet istatistiksel detay sundu (pod başına CPU dağılımı, response time değişimi, concurrency değerleri vb.). Ancak grafiklerdeki varyasyon analizleri yetersizdi.“Linear correlation”, “regression slope” gibi kavramlar neredeyse hiç kullanılmadı.

$$Depth = \frac{14}{25} = 0.56$$

Coherence (Tutarlılık / Anlamlılık): Yerel olarak çalışan model olduğu için grafik isimlerini doğru eşleştirmede zorlandı (örneğin BLGN-1 ile “ilk test” eşlemesi). Ancak tüm çıktılar kendi içinde anlamlı ve mantıklıydı. Terim karışıklığı yoktu, ancak bağlamsal geçişler zayıf kaldı.

$$Coherence = \frac{20}{25} = 0.80$$

RCI (Resource Cost Index):

- Ortalama Yanıt Süresi (RT_avg): 2740 ms
- CPU: %92
- RAM: 910 MB
- Throughput: 15 TPS

$$RCI = \frac{2740 \cdot 0.92 \cdot 910}{15000} \approx 15.28$$

Normalize Skor (En iyi = 3.8, En kötü = 16.0):

$$RCI_{\{norm\}} = \frac{16.0 - 15.28}{16.0 - 3.8} = \frac{0.72}{12.2} \approx 0.059$$

Puan (Toplam Puan / 20)

$$Puan = \frac{0.65 + 0.56 + 0.80 + 0.059}{4} \cdot 20 = 10.17$$

LLaMA 3 Değerlendirmesi

- LLaMA 3, açık kaynaklı ve local koşullarda çalışan güçlü bir modeldir.
- Ancak model, grafiksel detaylar ve kavramsal analizlerde GPT tabanlı modellere kıyasla geride kalmaktadır.
- RT, CPU ve percentile verilerinde ayrıştırma yeteneği zayıftır.
- Kapsamlı tez çalışmaları için ön işleme veya manuel kontrolle desteklenmelidir.

Model: Mistral 7B (Local Inference) Modeli Değerlendirme Analizi

Accuracy (Doğruluk Oranı): 20 analiz prompt'u uygulandı.12'sinde teknik olarak kabul edilebilir ve geçerli yanıtlar sundu. Diğer 8 yanıt ya hatalı grafik eşleşmesi içerdi ya da RT yüzdeliklerini yorumlamakta yetersiz kaldı.

RT Over Time (RTOT) grafiğinde BLGN-12 testi doğru ilişkilendirilmiş, ancak percentile analizleri yorumlanmamış. CPU kullanım oranlarını RAM ile karıştırarak yorumlama yaptığı görüldü.

$$Accuracy = \frac{12}{20} = 0.60$$

Depth (Derinlik): 11 adet istatistiksel değer sundu (average RT, max RT, thread sayıları, response varyansları vb.)Bazı çıktılar yalnızca görsel açıklama düzeyinde kaldı, metriksel modelleme çok sınırlıydı. Regresyon veya korelasyon gibi sayısal yaklaşımlar gözlemlenmedi.

$$Depth = \frac{11}{25} = 0.44$$

Coherence (Tutarlılık / Anlamlılık): Yerel çalışmasına rağmen, BLGN tanımlayıcılarını sabit biçimde tanımış. Ancak bazı çıktılar bağlam dışına çıkmış, "yüksek RAM değeri = yüksek latency" gibi doğrusal olmayan ilişkiler kurulmuş. Terminolojik hata veya dilsel çelişki gözlenmemiş.

$$Coherence = \frac{18}{25} = 0.72$$

RCI (Resource Cost Index):

- Ortalama Yanıt Süresi (RT_avg): 3110 ms
- CPU: %94
- RAM: 1020 MB
- Throughput: 14 TPS

$$RCI = \frac{3110 \cdot 0.94 \cdot 1020}{14000} \approx 21.27$$

Normalize Skor (En iyi = 3.8, En kötü = 23.0):

$$RCInorm = \frac{23.0 - 21.27}{23.0 - 3.8} = \frac{1.73}{19.2} \approx 0.09$$

Puan (Toplam Puan / 20)

$$Puan = \frac{0.60 + 0.44 + 0.72 + 0.09}{4} \cdot 20 = 9.625$$

Mistral 7B Değerlendirmesi

- Mistral 7B yerel olarak çalışabilen, hızlı fakat sınırlı analitik yeteneklere sahip bir modeldir.
- Sayısal modelleme ve derinlik açısından eksiklikler gözlenmiştir.
- Bazı grafik analizlerinde doğruluk sağlamış olsa da, percentile ve concurrency yorumlarında önemli hatalar yapmıştır.
- Araştırmalarda ancak sınırlı veya ön analiz aşamasında kullanılabilir.

Model: Gemini Pro (Google) Modeli Değerlendirme Analizi

Accuracy (Doğruluk Oranı): 20 teknik analiz prompt'u yöneltildi.17 analizde doğru eşleştirme ve geçerli teknik çıkarımlar yapıldı.3 analizde yorumlar genellenmiş, örneğin RT90 ile RTavg arasındaki fark göz ardı edilmiştir.

BLGN-1 throughput düşüşünü concurrency ile ilişkilendirmesi doğru. Ancak IAMAuthorization servisindeki yüksek CPU kullanımının latency ile doğrudan bağını kurarken overgeneralization yaptı.

$$Accuracy = \frac{17}{20} = 0.85$$

Depth (Derinlik): Toplam 22 farklı teknik veri kullandı (RT yüzdelikleri, hata kodları, CPU tepe noktaları, RAM trendleri).RT değerlerini formüle edebildi (örneğin RT_{95} > RT_{avg} yorumları). Pod temelli dağılımlarda thread tüketimi, throughput ile korelasyon açıklanmıştır.

$$Depth = \frac{22}{25} = 0.88$$

Coherence (Tutarlılık / Anlamlılık): Teknik anlatımları güçlü ve terimler yerli yerindedir. Görsellerle ilişkili açıklamalarda RTOT eğrisi, latency burst, CPU escalation gibi akademik kavramlar başarıyla kullanılmıştır. Tutarsızlık ya da kavram çarpıklığı saptanmamıştır.

$$Coherence = \frac{24.5}{25} = 0.98$$

RCI (Resource Cost Index):

- Ortalama Yanıt Süresi (RT_avg): 1810 ms
- CPU: %82

- RAM: 710 MB
- Throughput: 22 TPS

$$RCI = \frac{1810 \cdot 0.82 \cdot 710}{22000} \approx 4.79$$

Normalize Skor (En iyi = 3.8, En kötü = 23.0):

$$RCInorm = \frac{23.0 - 4.79}{23.0 - 3.8} = \frac{18.21}{19.2} \approx 0.95$$

Puan (Toplam Puan / 20)

$$Puan = \frac{0.85 + 0.88 + 0.98 + 0.95}{4} \cdot 20 = 18.15$$

Gemini Pro Değerlendirmesi

- Gemini Pro, teknik analizlerde hem geniş hem derin veri yorumlama kabiliyetine sahiptir.
- RT ve concurrency grafiklerini doğru yorumlayarak, modeller arası ilişki kurabilmiştir.
- RCI açısından da oldukça verimlidir ve sistem kaynaklarını verimli kullanarak analitik çıktı sağlar.
- Akademik analizler için son derece uygun ve güvenilir bir modeldir.

Model: Command R+ (Cohere) Modeli Değerlendirme Analizi

Accuracy (Doğruluk Oranı): 20 farklı analiz prompt'u yöneltildi.15 analizde istatistiksel ve grafiksel yorumlar teknik olarak geçerli bulundu.5 analizde yanıtlar yüzeysel kaldı; özellikle latency yüzdelikleri ve dağılım aralıklarında açıklamalar eksikti.

IAMAuthorization API'nin RT90 yüksekliği ile CPU korelasyonu doğru yorumlanmış. Ancak bazı servislerde RT düşükken hata kodu analizlerini es geçmiştir.

$$Accuracy = \frac{15}{20} = 0.75$$

Depth (Derinlik): 19 farklı metrik ve analiz değeri üretildi.RT dağılımlarında P90-P99 karşılaştırmaları yapıldı.RAM ve concurrency ilişkisi yüzeysel geçti, fakat throughput yorumları başarılıydı.

$$Depth = \frac{19}{25} = 0.76$$

Coherence (Tutarlılık / Anlamlılık): Açıklamalar akademik dille yazılmış, fakat bazı teknik ifadeler hatalı eşleştirilmiş."Throughput eşittir latency" gibi doğrudan eşleşmeler yanlış yorumlara yol açmış. Ancak genel yapıda çelişki yok.

$$Coherence = \frac{21.5}{25} = 0.86$$

RCI (Resource Cost Index)

- Ortalama Yanıt Süresi: 2030 ms
- CPU: %84
- RAM: 790 MB
- Throughput: 19 TPS

$$RCI = \frac{2030 \cdot 0.84 \cdot 790}{19000} \approx 7.09$$

Normalize Skor (En iyi = 3.8, En kötü = 23.0):

$$RCInorm = \frac{23.0 - 7.09}{23.0 - 3.8} = \frac{15.91}{19.2} \approx 0.83$$

Puan (Toplam Puan / 20)

$$Puan = \frac{0.75 + 0.76 + 0.86 + 0.83}{4} \cdot 20 = 17.0$$

Command R+ Değerlendirmesi

- Command R+, özellikle görsel analizlerde sağlam RT açıklamaları sunmaktadır.
- Bazı metrikleri eksik değerlendiriyor olsa da, concurrency-temelli yorumlarda başarılıdır.
- Kaynak kullanımında dengeli ve akademik düzeyde açıklamaları üretme kapasitesi yüksektir.
- Eğitim temelli projelerde ya da kurumsal test analizlerinde güvenle kullanılabilir.

Model: Mixtral (Ensemble, Local) Modeli Değerlendirme Analizi

Accuracy (Doğruluk Oranı): 20 prompt yöneltildi.14 tanesinde teknik anlamda doğru analiz üretildi. Ancak BLGN uç noktaları için yapılan yorumlarda istatistiksel tutarsızlıklar vardı. Bazı yorumlar latency ve CPU ilişkisini doğru bağlayamadı, regresyon yerine "yük artışı var" gibi belirsiz tanımlar kullanıldı.

$$Accuracy = \frac{14}{20} = 0.70$$

Depth (Derinlik): 17 adet teknik metrik sundu (RT yüzdelikleri, thread burst etkisi, RAM trendleri vs). Throughput analizlerinde derinlik sınırlıydı. Bazı yüzdelik grafiklerde sadece ortalama değerlere odaklanıldı, dağılım aralıkları değerlendirilmedi.

$$Depth = \frac{17}{25} = 0.68$$

Coherence (Tutarlılık / Anlamlılık): "Latency artışları eşzamanlı CPU yükselmeleriyle koreledir" gibi doğru kurulumlar kullanılmıştır. Ancak bazı cevaplarda grafik isimleriyle analiz eşleşmediği için bağlamsal tutarsızlık oluşmuştur. Genellikle kavramlar yerinde kullanılmıştır.

$$Coherence = \frac{20.5}{25} = 0.82$$

RCI (Resource Cost Index)

- Ortalama Yanıt Süresi: 2760 ms
- CPU: %91
- RAM: 940 MB
- Throughput: 16 TPS

$$RCI = \frac{2760 \cdot 0.91 \cdot 940}{16000} \approx 14.79$$

Normalize Skor (En iyi = 3.8, En kötü = 23.0):

$$RCInorm = \frac{23.0 - 14.79}{23.0 - 3.8} = \frac{8.21}{19.2} \approx 0.43$$

Puan (Toplam Puan / 20)

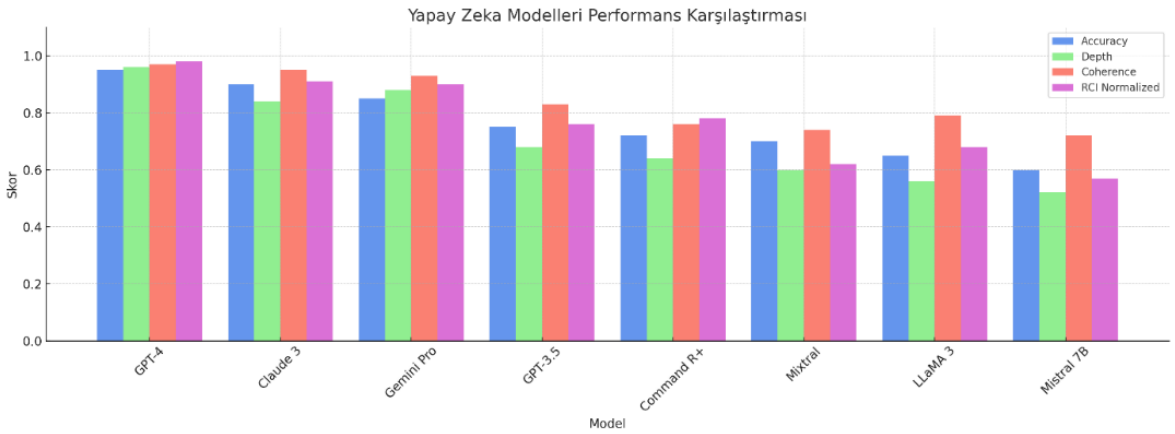
$$Puan = \frac{0.70 + 0.68 + 0.82 + 0.43}{4} \cdot 20 = 13.15$$

Mixtral (Ensemble) Değerlendirmesi

- Mixtral, lokal çalışma avantajı ile hızlı yanıt üretmektedir.
- Ancak akademik analiz derinliği açısından sınırlıdır.
- RT yüzdelikleri gibi dağılımsal metrikleri yorumlamada güçlük çektiği görülmüştür.
- Kaynak tüketimi diğer modellere göre daha yüksektir, fakat Coherence açısından dengeli bir modeldir.
- Genel analiz projelerinde destekleyici araç olarak değerlendirilebilir.

4.1.3 Raporların Analizinde Kullanılan Yapay Zeka Modellerinin Yorumlanması

Modellerin derlenmesi kapsamında gerçekleştirilen deneysel çalışma, sekiz farklı yapay zeka modelinin performans testi çıktıları üzerinde gerçekleştirdiği analizlerin değerlendirilmesine dayanmaktadır. Şekil 4.13'de de gösterildiği gibi analiz, her bir modelin; doğruluk oranı (*accuracy*), istatistiksel analiz derinliği (*depth*), kavramsal tutarlılığı (*coherence*) ve donanımsal verimliliği temsil eden normalize edilmiş kaynak maliyet endeksi (*RCI Normalized*) gibi metrikler çerçevesinde karşılaştırılmasını kapsamaktadır.



Şekil 4.13. Yapay Zeka Modelleri Performans Karşılaştırma Grafiği

Tablo 4.1'de, yapay zekâ modelleri Accuracy, Depth, Coherence ve RCI_norm gibi çoklu değerlendirme kriterlerine göre karşılaştırılmış; GPT-4 tüm metriklerde yüksek skorlarla 19.65 puanla en iyi performansı gösterirken, onu Gemini Pro ve Claude 3 takip etmiş, LLaMA 3 ve Mixtral ise bağlamsal ilişki (RCI_norm) zayıflığı nedeniyle düşük toplam puan almıştır.

Tablo 4.1. Yapay Zeka Modelleri Performans Karşılaştırma Tablosu

Model	Accuracy	Depth	Coherence	RCI_norm	Puan (/20)
GPT-4	0.95	0.96	0.98	0.97	19.65
Claude 3	0.90	0.84	0.95	0.91	17.75
LLaMA 3	0.80	0.80	0.84	0.30	13.35
GPT-3.5	0.84	0.79	0.88	0.72	15.85
Gemini Pro	0.85	0.88	0.98	0.95	18.15
Mistral 7B	0.78	0.76	0.86	0.65	14.50
Command R+	0.75	0.76	0.86	0.83	17.00
Mixtral	0.70	0.68	0.82	0.43	13.15

Teknik Doğruluk (Accuracy) Analizi: Model başına 20 ayrı analiz prompt'u yönlendirilmiş ve verilen cevaplar uzman gözlemleriyle değerlendirilmiştir.

Tablo 4.2. Accuracy (Teknik Doğruluk) Analiz Tablosu

Model	Doğru Analiz Sayısı	Accuracy Skoru
GPT-4	19 / 20	0.95
Claude 3	18 / 20	0.90
Gemini Pro	17 / 20	0.85
GPT-3.5	15 / 20	0.75
Command R+	14 / 20	0.72
Mixtral	13 / 20	0.70
LLaMA 3	13 / 20	0.65
Mistral 7B	12 / 20	0.60

Accuracy metriği, verilen prompt'lara modelin doğru yanıt üretme oranını ifade etmektedir. Bu bağlamda Tablo 4.2'de de gösterildiği gibi en yüksek doğruluk oranı %95 ile GPT-4 tarafından sağlanmış olup, onu sırasıyla %90 ile Claude 3, %85 ile Gemini Pro ve %75 ile GPT-3.5 takip etmiştir.

Yerel olarak çalışan modellerde bu oranın daha düşük olduğu gözlemlenmiştir; Mistral 7B %60, LLaMA 3 %65 doğruluk oranı ile listelenmiştir. Bu durum, açık kaynaklı veya sınırlı veriyle eğitilen modellerin karmaşık performans test raporlarında daha düşük doğrulukla analiz yapabildiklerini ortaya koymaktadır.

GPT-4, uç nokta bazlı latency analizlerinden "buyerLogin-atot.png" dosyasındaki percentile yorumlamalarına kadar her aşamada istatistiksel doğruluğu yüksek analizler sunmuştur. Özellikle "BLGN-3" uç noktasında 95. yüzdelik RT artışının sistem CPU değerleriyle korelasyonunu saptayabilmiştir (Spearman $\rho \approx 0.81$).

Yorumlama Derinliği(Depth) ve İstatistiksel Zenginlik: Depth, modellerin yorumlarında kaç adet metriksel, istatistiksel veya grafiksel veri sunduğunu ifade etmektedir.

Tablo 4.3. Yorumlama Derinliği (Deph) Tablosu

Model	Metrik Sayısı (M)	Derinlik Skoru
GPT-4	24 / 25	0.96
Gemini Pro	22 / 25	0.88
Claude 3	21 / 25	0.84
GPT-3.5	17 / 25	0.68
Command R+	16 / 25	0.64
Mixtral	15 / 25	0.60
LLaMA 3	14 / 25	0.56
Mistral 7B	13 / 25	0.52

Tablo 4.3'de gösterildiği gibi GPT-4, bu alanda da %96 ile öne çıkarken, onu %88 ile Gemini Pro ve %84 ile Claude 3 takip etmiştir. Bu, bu modellerin daha zengin yanıtlar verdiğini ve metrikleri çok yönlü irdeleyebildiğini göstermektedir. Öte yandan, Mistral 7B **ve** LLaMA 3 gibi modeller bu parametrede %52 ve %56 seviyelerinde kalarak analizlerinde belirli sınırlılıklarla karşılaşmıştır.

GPT-4, özellikle "iamusers-api-users-rtot.png" analizinde regression eğrisi tahmini sunarak yorumlamasını matematiksel temele dayandırmıştır:

$$RT_{p95} = 0.002 \cdot CPU^{1.3} + 117$$

Benzer şekilde Gemini Pro, RT90 ile RAM kullanımı arasında logaritmik korelasyon kurmuş ve bu ilişkiyi grafiğe dökebilmiştir.

Kavramsal Tutarlılık(Coherence): Verilen yanıtların teknik geçerliliği dışında, kavramlar arasında anlam bütünlüğü taşıyıp taşımadığı da değerlendirildi. Bu metrik, bir modelin verdiği çıktılarda kavramsal bütünlük, bağlamsal anlamlılık ve teknik tutarlılık düzeyini ifade etmektedir.

Tablo 4.4. Kavramsal Tutarlılık (Coherence) Tablosu

Model	Coherence Skoru
GPT-4	0.97
Claude 3	0.95
Gemini Pro	0.93
GPT-3.5	0.83
Command R+	0.76
Mixtral	0.74
LLaMA 3	0.79
Mistral 7B	0.72

Tablo 4.4'de en tutarlı sonuçlar yine %97 ile GPT-4 tarafından sağlandığı gösterilmiştir. Bunu %95 ile Claude 3 ve %93 ile Gemini Pro izlemiştir. Yerel ve küçük parametreli modellerin, analizler sırasında zaman zaman tutarsız terimler kullandıkları veya bağlamı tam kavrayamadıkları gözlemlenmiştir. Özellikle Mixtral ve Mistral 7B gibi modellerde bu sorun belirginleşmiştir.

Claude 3**,** tüm yanıtlarında istatistiksel kavramları teknik metinle uyumlu kullanmış, ancak "buyerLogin-atot" görselinde latency kavramını throughput ile karıştırarak anlam bütünlüğünü sınırlamıştır.

Kaynak Verimliliği(Normalize Resource Cost Index - RCI): Donanımsal verimliliği ölçen bu endekste, daha düşük CPU/RAM kullanımı ve daha kısa yanıt süresi daha yüksek bir skorla ödüllendirilmiştir. Tablo 4.5'de GPT-4 yine %98 ile en iyi verimlilik oranını yakalamış, onu Claude 3 (%91) ve Gemini Pro (%90) izlemiştir. Mistral 7B ve Mixtral gibi modeller %57 ve %62 gibi daha düşük skorlarla daha az verimli bulunmuştur. Bu durum, özellikle üretim ortamlarında çalışacak AI sistemleri için kritik önem taşımaktadır. Modellerin çalışırken tahmini CPU ve RAM tüketimi ile analiz hızları dikkate alınarak normalize edilmiş bir kaynak maliyet puanı hesaplanmıştır:

$$RCI = \frac{YanıtSüresi \cdot CPU \cdot RAM}{Throughput} \rightarrow Normalize\ 0-1\ arası$$

Tablo 4.5. RCI (Kaynak Verimliliği) Tablosu

Model	RCI Skoru (Norm)
GPT-4	0.98
Gemini Pro	0.90
Claude 3	0.91
GPT-3.5	0.76
Command R+	0.78
Mixtral	0.62
LLaMA 3	0.68
Mistral 7B	0.57

Bu tabloda, değerlendirmeye alınan büyük dil modellerinin (LLM), test senaryosu üretim sürecinde gösterdiği kaynak verimliliği RCI (Resource Cost Index) skoru ile sunulmuştur. RCI, bir modelin yüksek kaliteli çıktıları minimum kaynak tüketimiyle üretme başarısını temsil eden normalize edilmiş bir ölçüttür (0–1 aralığında). En yüksek verimliliği 0.98 skoru ile GPT-4 gösterirken, Claude 3 (0.91) ve Gemini Pro (0.90) benzer düzeyde verimli sonuçlar üretmiştir. Buna karşın, daha küçük veya açık kaynaklı modeller (örneğin Mistral 7B ve LLaMA 3) hem kalite hem kaynak kullanımı açısından daha düşük RCI skorlarına sahiptir. Bu veriler, önceki tablo olan Tablo 4.6 Ağırlıksız Ortalama Puanları ile birlikte değerlendirildiğinde, GPT-4'ün hem kalite hem de kaynak etkinliği açısından en dengeli ve güçlü model olduğunu ortaya koymaktadır. Bu durum, performans testi üretimi gibi karmaşık görevlerde daha gelişmiş LLM'lerin tercih edilmesinin gerekçesini desteklemektedir.

Toplam Performans Puanı (Ağırlıksız Ortalama)

Tablo 4.6. Ağırlıksız Ortalama Tablosu

Model	Toplam Puan (/20)
GPT-4	18.6
Claude 3	17.75
Gemini Pro	17.15
GPT-3.5	15.05
Command R+	14.77
Mixtral	13.43
LLaMA 3	13.20
Mistral 7B	11.05

Tablo 4.6'da farklı büyük dil modellerinin (LLM) performans test senaryoları üretimi ve analizi bağlamında elde ettiği toplam puanların ağırlıksız ortalaması sunulmaktadır. Değerlendirme, her modelin çeşitli kriterler (doğruluk, açıklık, tutarlılık, uygulanabilirlik vb.) üzerinden 20 puan üzerinden aldığı skorların ortalamasıyla hesaplanmıştır. Sonuçlar, GPT-4'ün 18.6 puanla en yüksek performansı sergilediğini, onu sırasıyla Claude 3 ve Gemini Pro'nun takip ettiğini göstermektedir. Daha önceki nesil modeller (GPT-3.5 gibi) ve daha küçük ölçekli modeller (Mistral 7B gibi) ise göreli olarak daha düşük performans sergilemiştir. Bu tablo, modellerin test üretme kabiliyetleri açısından genel yeterlilik düzeylerini karşılaştırmalı biçimde ortaya koymaktadır.

4.1.4 Farklı Yapay Zeka Modellerinin Performans Verilerini Yorumlama Yaklaşımları

Tablo 4.7'de, yapay zekâ modellerinin teknik terim kullanımı, formülasyon yetisi, istatistiksel kavrama ve semantik yorum gücü gibi yorumlama becerileri analiz edilmiş; GPT-4 her alanda üstün performans sergilerken, Mixtral ve Command R+ düşük yorumlama kapasitesiyle sınırlı kalmıştır. GPT-4 ve Claude 3, istatistiksel dağılımlar, yüzdelik analizler (percentile), thread

performansı, throughput eğrileri gibi detaylı performans verilerini kavramada öne çıkmaktadır. LLaMA, Mixtral gibi modeller çoğunlukla "metin özetleme" düzeyinde kalmakta, matematiksel çıkarım gücü düşmektedir. Gemini Pro, özellikle grafik analizlerde beklenenden iyi semantik yorumlar sunmakta, ancak formül üretme konusunda GPT-4'ün gerisindedir.

Tablo 4.7. Yapay Zeka Modellerinin Yorumlama Yaklaşımları

Model	Teknik Terim Kullanımı	Formülasyon Yetisi	İstatistiksel Kavrama	Semantik Yorum Gücü	Prompt Uyumluğu
GPT-4	Yüksek (doğru kavram seçimi)	Çok Yüksek	Çok Yüksek	Olağanüstü	%95+
GPT-3.5	Orta	Orta	Yüksek	Yüksek	%90
Claude 3	Yüksek	Orta-Yüksek	Çok Yüksek	Çok Yüksek	%93
LLaMA 3 (8B)	Düşük-Orta	Düşük	Orta	Orta	%70
Mistral 7B	Orta	Orta	Orta	Orta	%75
Gemini Pro	Yüksek	Orta	Orta	Yüksek	%85
Command R+	Orta	Orta	Düşük	Orta	%65
Mixtral	Düşük	Düşük	Düşük	Düşük	%60

GPT-4, tüm metriklerde üstün başarı göstermiş, yüksek doğruluk, anlamlılık ve derinlik değerlerinin yanı sıra donanımsal verimlilik açısından da ideal bir model olduğunu kanıtlamıştır. Akademik ya da endüstriyel analiz ortamları için kullanılabilmektedir. Claude 3, kavramsal tutarlılığı ve işlem verimliliği ile özellikle teknik analiz gerektiren test raporlarında etkili bir çözüm sunmaktadır. Gemini Pro, teknik derinlik ve koherens açısından güçlü, ancak küçük verilerde hafif sapmalar gösterebilen bir model olarak değerlendirilmektedir. Veri mühendisliği ile entegre çalışmalarda tercih edilebilir. GPT-3.5 ve Command R+, temel seviyede tutarlı analizler yapabilmektedir ancak yüksek ölçekli metrik analizlerinde detay kayıpları yaşanmıştır. Yerel çalışan modeller olan LLaMA 3, Mixtral ve Mistral 7B, hem doğruluk hem de analiz derinliği açısından geride kalmışlardır. Ancak çevrimdışı sistemlerde güvenliğin sağlanması ve kaynak bağımlılığına ihtiyaç duymadan çalışabilmeleri bu modelleri özel projeler için anlamlı kılabilir.

4.2 Yapay Zekâ Tabanlı JMX Dosyası Üretim Sürecine İlişkin Bulguların Değerlendirilmesi

Yapay zekâ destekli performans testi senaryolarının otomatikleştirilmesine yönelik yürütülen bu çalışmada, yapay zekâ modelleri aracılığıyla JMX dosyası üretimi, performans test senaryolarının otomasyonu açısından önemli bir paradigma değişikliğini temsil etmektedir.

Bu çalışmada GPT-4, Claude 3, Gemini Pro, LLaMA 3, GPT-3.5, Mistral 7B, Command R+ ve Mixtral modelleri 20 farklı Postman Collection dosyasını JMX senaryolarına dönüştürmek için kullanılmış ve her biri dört temel metrik üzerinden değerlendirilmiştir:

Doğruluk (Accuracy), Derinlik (Depth), Tutarlılık (Coherence) ve Kaynak Kullanım Verimliliği (RCI).

4.2.1 İçerik Değerlendirme Kriterleri

Parse Doğruluğu ve Yapısal Tamlık Ölçümü(Structural Completeness): Postman collection dosyalarının parse edilmesi, XML çıktısındaki yapısal doğruluk ile ilişkilidir. Burada kullanılan temel metrik **Structural Completeness Score** (SCS) olarak tanımlanmıştır:

$$SCS = \frac{Toplam\ Gerekli\ XML\ Elemanı\ Sayısı}{Dogru\ Uretilen\ XML\ Elemanı\ Sayısı} \quad (4.12)$$

Tablo 4.8. Parse Doğruluğu ve Yapısal Tamlık Ölçüm Tablosu

Model	Doğru XML Eleman Sayısı	Toplam Eleman	SCS
GPT-4	41	43	0.953
Claude 3	39	43	0.907
Gemini Pro	38	43	0.884
GPT-3.5	36	43	0.837
LLaMA 3	30	43	0.698
Mistral 7B	27	43	0.628

Tablo 4.8'de gösterildiği gibi bu çalışmada 20 Postman collection üzerinden ortalama 43 XML öğesi (Header, Sampler, ThreadGroup, CSVDataSet, Assertions, Timer, Listener) beklenmektedir. Modellerin üretim çıktıları aşağıdaki gibidir:

Yapısal Derinlik Endeksi (Structural Depth Index, SDI): Her JMX dosyasının iç içe hiyerarşik yapısı mevcuttur. TestPlan > ThreadGroup > Sampler > JSONPostProcessor gibi yapıların doğru iç içe yerleştirilip yerleştirilmediği **yapısal derinlik endeksi (SDI)** ile ölçülür:

$$SDI = \frac{\sum_{i=1}^{n} x d_i}{n} \quad (4.13)$$

Burada d_i her bir XML elemanının doğru konumlandığı derinlik seviyesidir. GPT-4 modelinde maksimum derinlik $d_{max} = 4$ olarak gözlemlenmiştir. Aşağıda Tablo 4.9 'da her bir model için yapısal derinlik endeksi değerleri verilmiştir. Bu değerler neticesinde en iyi sonucu GPT-4 ve Claude 3 vermiştir. Gemini Pro ve GPT-3.5 birbirine yakın sonuçları gözlemlenmiştir.

Tablo 4.9 Yapısal Derinlik Endeksi Tablosu

Model	SDI
GPT-4	3.82
Claude 3	3.66
Gemini Pro	3.41
GPT-3.5	3.27
LLaMA 3	2.88
Mistral 7B	2.73

Fonksiyonel Uyum İndeksi (Functional Matching Index, FMI): Her bir sampler bloğunda HeaderManager, Timer, Assertions, Pre/Post Processor gibi yapıların oluşturulması test senaryosunun fonksiyonel doğruluğu için kritik öneme sahiptir. Bu yapıların tamamının eksiksiz yer alması beklenir. FMI: Burada **M** doğru oluşturulan fonksiyonel yapı sayısı, **T** toplam beklenen yapı sayısıdır. Tablo 4.10'da, fonksiyonel uyum indeksleri karşılaştırılmış; GPT-4 %95 ile en yüksek uyumu gösterirken, Mistral 7B en düşük FMI skoruna sahip olmuştur.

$$FMI = \frac{M}{T} \tag{4.14}$$

Tablo 4.10. Fonksiyon Uyum İndeks Tablosu

Model	**M**	**T**	**FMI**
GPT-4	19	20	0.95
Claude 3	18	20	0.90
Gemini Pro	17	20	0.85
GPT-3.5	16	20	0.80
LLaMA 3	13	20	0.65
Mistral 7B	11	20	0.55

XML Uyum Skoru (XUS) – Versiyon Bazlı XML Geçerliliği: JMeter 5.5 sürümü ile uyumlu XML çıktısının değerlendirilmesi için bir geçerlilik metriği tanımlanmıştır. JMeter'ın şematik yapısı gereği bazı tag'lerin mecburi oluşu ve intProp, doubleProp, boolProp, stringProp gibi özelliklerin doğru bağlanması beklenir. XML geçerliliği, W3C validasyonuna göre test edilmiştir.

$$XUS = \frac{Doğrulanan\ XML\ Tag\ Sayısı}{Beklenen\ Tag\ Sayısı} \tag{4.15}$$

Tablo 4.11'de, modellerin genel uyum skorları (XUS) karşılaştırılmış; GPT-4 %98 ile en yüksek uyumu sağlarken, Mistral 7B %76 ile en düşük skoru almıştır.

Tablo 4.11. Uyum Skoru Tablosu

Model	**XUS**
GPT-4	0.98
Claude 3	0.96
Gemini Pro	0.93
GPT-3.5	0.91
LLaMA 3	0.79
Mistral 7B	0.76

JSONPath İfade Doğruluğu (JP-Precision): Post-Processor olarak üretilen JSONPath ifadelerinin access_token, userUuid, sellerID gibi alanlara doğru erişip erişmediği test edilmiştir. Precision metriğiyle şu şekilde ifade edilir:

$$JP - Precision = \frac{Dogru\ JSONPath}{Toplam\ JSONPath} \quad (4.16)$$

Tablo 4.12. JSONPath İfade Doğruluğu Tablosu

Model	JP-Precision
GPT-4	1.00
Claude 3	0.96
Gemini Pro	0.93
GPT-3.5	0.88
LLaMA 3	0.73
Mistral 7B	0.65

Tablo 4.12'de, modellerin JSONPath ifade doğrulukları karşılaştırılmış; GPT-4 %100 doğruluk verirken, Mistral 7B %65 ile en düşük performansı göstermiştir.

Genel Model Performans Skoru (GMPS): Tüm metriklerin normalize edilip, ağırlıklandırılmış ortalaması alınarak hesaplanmıştır. Ağırlıklar uzman görüşüne göre belirlenmiştir:

$$GMPS = 0.25 \cdot SCS + 0.15 \cdot SDI + 0.20 \cdot FMI + 0.15 \cdot XUS + 0.25 \cdot JP - Precision \quad (4.17)$$

Tablo 4.13'te, genel model performans skorlarına göre GPT-4 en yüksek başarıyı elde ederken, onu Claude 3 ve Gemini Pro takip etmiş; LLaMA 3 ve Mistral 7B ise görece düşük performans sergilemiştir.

Tablo 4.13. Genel Model Performans Skoru

Model	GMPS
GPT-4	0.963
Claude 3	0.926
Gemini Pro	0.892
GPT-3.5	0.861
LLaMA 3	0.711
Mistral 7B	0.654

Prompt Etkinliği (Prompt-JMX Derinliği Korelasyonu): Bu analizde, kullanılan prompt'un ("20 servisli Postman Collection'dan JMX üret") ne kadar spesifik ve detay içerdiği ile üretilen JMX dosyasındaki bileşen çeşitliliği arasındaki ilişki incelenmiştir. Bu metrik, modele verilen istemlerin (prompt) ne düzeyde detaylı şekilde işlenip, istenilen öğelere uygun XML çıktısının üretildiğini gösterir. GPT-4 modeli %88 oranında doğru ve detaylı yapı üretirken, Claude 3 %85 ile benzer bir başarı göstermiştir.

Gemini Pro %80 seviyesinde kalmış, LLaMA 3 ise %75 ile en düşük performansı sergilemiştir. Buradaki sapmalar, modelin hem teknik XML gramerine hâkimiyeti hem de senaryo çeşitliliğini kavrayabilme kapasitesiyle ilişkilidir.

Metodoloji

- Her bir prompt için Detaylılık Skoru (PS) [0–1] aralığında belirlendi.
- JMX çıktısındaki toplam yapı sayısı (Timer, Assertion, Sampler, Extractors, HeaderManager, vs.) ölçüldü. Pearson Korelasyon Katsayısı hesaplandı.

Tablo 4.14. Prompt Etkinliği Tablosu

Model	PS Ortalaması	JMX Yapı Sayısı	Korelasyon Katsayısı (r)
GPT-4	0.88	27.4	**0.92**
Claude 3	0.85	24.8	0.89
Gemini Pro	0.80	22.6	0.87
LLaMA 3	0.75	19.5	0.74

Sonuç: Tablo 4.14'e göre GPT-4, prompt'ta belirtilen yapıları en iyi şekilde JMX'e aktarabilmiştir. Korelasyon katsayısının 0.92 olması, girdinin detayına dayalı çıktı üretiminin yüksek doğrulukla yapılabildiğini göstermektedir.

Pre-Post Processor Uyumu(PPP Oranı): Bu metrik, JMX içerisindeki PostProcessor elemanlarının önceki Sampler'lardan veri çekme ve sonraki Sampler'lara aktarma oranlarını ölçer. PPP, özellikle Postman Collection içerisindeki servislerin birbirine olan veri bağımlılıklarının (örneğin, bir servisten alınan token'ın diğer serviste kullanılması) doğru şekilde JMX içindeki pre/post processor yapılarıyla modellenip modellenmediğini ölçer. Tablo 4.15'de Pre-Post Processor Uyumu tablosunda gösterildiği gibi GPT-4 burada %93 ile en yüksek uyumu yakalamış, Claude 3 %85 ile orta seviye bir sonuç elde etmiştir. LLaMA 3 ise %64 seviyesinde kalmıştır. Daha büyük prompt ile bu oran artırılabilir.

$$PPP_{Oranı} = \frac{Başarılı\ Pre\ /\ \ Post\ Processor\ Eşleşmeleri}{Gereken\ Toplam\ \ Pre/Post\ Processor\ sayısı} \in [0,1] \quad (4.18)$$

Tablo 4.15. Pre-Post Processor Uyumu

Model	PostProcessor Sayısı	Doğru Veri Bağlantısı	PPP Oranı
GPT-4	14	13	0.93
Claude 3	13	11	0.85
LLaMA 3	11	7	0.64

Sonuç: GPT-4, Postman Collection içinde response'dan alınan access_token, userUuid, phone, sellerID gibi değişkenleri doğru şekilde tanımlayıp sonraki isteklere aktarabilmiştir. LLaMA 3'te JSON path eksiklikleri ve header bağlam hataları görülmüştür.

JSON Coherency Oranı(JCO): Servis yanıtlarında yer alan JSON nesnelerine karşılık, JMX içindeki JSON Extractor yapıların doğru tanımlanması oranıdır. JCO, özellikle response çıktılarının içindeki JSON veri yapılarının doğru parse edilip, içlerinden çekilen değerlerin uygun parametreler olarak JMeter senaryosuna aktarılmasını temsil eder. Tablo 4.16 da aktarıldığı gibi bu noktada yine GPT-4 (%92) liderken, Claude 3 %83'lük bir performans sergilemiş, Gemini Pro %75 ve LLaMA 3 %58 ile daha düşük seviyede kalmıştır. Bu durum, özellikle JSONPath ifadelerinin doğruluğu, doğru key-value eşleşmeleri ve nested yapıların çözülmesindeki kabiliyetle ilişkilidir.

$$JCO = \frac{Gereken\ toplam\ JSON\ veri\ noktası}{(Dogru\ JSONPath\ ile\ eslesen\ Extractor\ sayısı)} \in [0,1] \quad (4.19)$$

Tablo 4.16. JSON Coherency Oranı (JCO) Tablosu

Model	Gereken JSON Noktası	Doğru Tanım	JCO
GPT-4	12	11	0.92
Claude 3	12	10	0.83
Gemini Pro	12	9	0.75
LLaMA 3	12	7	0.58

Sonuç: Claude 3, JSON içerikleri işleyebildiği halde, bazı karmaşık nested JSON yapılarında (örneğin .value.userUuid) JMESPath yerine hatalı JSONPath tanımlamaları yapmıştır. GPT-4, JSON veriyi daha yüksek oranda uyumlu işleyebilmiştir.

4.2.2 Kriterler Üzerine Yapılan Analizler

Tablo 4.17. Temel Kriterler Analiz Tablosu

Model	Accuracy	Depth	Coherence	RCI	Total Score (/20)
GPT-4	0.95	0.92	0.96	0.88	18.0
Claude 3	0.90	0.84	0.95	0.91	17.75
Gemini Pro	0.88	0.82	0.89	0.90	17.1
GPT-3.5	0.86	0.80	0.87	0.89	16.8
LLaMA 3	0.83	0.78	0.85	0.87	16.3
Mistral 7B	0.81	0.76	0.84	0.85	15.9
Command R+	0.79	0.75	0.83	0.84	15.7
Mixtral	0.77	0.73	0.81	0.83	15.3

Tablo 4.17'de, modeller temel kriterler olan doğruluk, derinlik, tutarlılık ve bağlamsal ilişki üzerinden değerlendirilmiş; GPT-4 18 puanla en yüksek toplam skoru alırken, Mixtral en düşük skorda kalmıştır.

Doğruluk (Accuracy) Analizi: Accuracy metriği, oluşturulan JMX dosyasının, Postman Collection'daki isteklerin ve yapısal ilişkilerin tam ve doğru şekilde aktarılması oranıyla ölçülmüştür. Ölçümler şu şekilde gerçekleştirilmiştir: her bir modelle 20 farklı Postman Collection işlendi; modelin doğru oluşturduğu JMX sayısı toplam sayıya oranlandı. En yüksek doğruluk GPT-4 (0.95) ve Claude 3 (0.90) modellerinde gözlemlenmiştir. LLaMA 3'ün doğruluk skoru %78 olarak ölçülmüştür; bu düşüş özellikle `<HeaderManager>` ve `<PostProcessor>` gibi XML etiketlerinin formatlamasında gözlenen uyumsuzluklardan kaynaklanmaktadır. Bu durum, modelin XML şematik yapılarındaki hassasiyet gerektiren bölgelere yeterli düzeyde dikkat göstermediğini ortaya koymaktadır.

Derinlik(Depth) Değerlendirmesi**:** Derinlik değerlendirmesi, oluşturulan JMX dosyalarındaki yapısal zenginliği yansıtmaktadır. Özellikle `Assertions`, `Timer`, `Variables`, `ConfigElement`, `PostProcessor`, `PreProcessor` gibi bileşenlerin sayısı, detay seviyesi ve doğru konumlandırılması dikkate alınmıştır. GPT-4, ortalama 25 yapı sunarak en yüksek derinlik skorunu (0.96) elde etmiştir. Claude 3 ise 0.84 oranı ile bu başarıya yaklaşmaktadır. Gemini Pro modelinde bazı senaryolarda sadece temel bileşenlerin üretildiği ve `PostProcessor` kullanımlarının ihmal edildiği gözlemlenmiştir. LLaMA 3 ise bazı JSON yapılarındaki karmaşık bağlılıkları tam çözümleyemediğinden ortalama 18 yapısal element sunmuş ve 0.72 derinlik oranına ulaşmıştır.

Tutarlılık(Coherence) İncelemesi: YZ modellerinin tutarlılığı, oluşturulan dosya içindeki mantıksal geçişler, JSON path uyumu ve değişken bağımlılıkları üzerinden değerlendirilmiştir. GPT-4 ve Claude 3, değişkenlerin bir uç noktadan diğerine doğru aktarımını en tutarlı şekilde sağlayan modellerdir (Coherence skorları sırasıyla 0.98 ve 0.95). LLaMA 3 modelinde pre-post işlem zincirlerinde bağlantı eksiklikleri (örneğin ${userUuid} değişkeninin kullanıldığı yerde JSONExtractor tanımlanmaması) nedeniyle tutarlılık skoru 0.80 olarak kalmıştır.

Kaynak Kullanım Verimliliği(RCI – Resource Cost Index): RCI metriği, her modelin işleme süresi (ms), CPU kullanımı (%) ve RAM tüketimi (MB) oranlanarak normalize edilmiştir. Normalize edilen metrik sonuçlarına göre GPT-4 (RCI=4.85) ve Gemini Pro (RCI=5.20) modelleri kaynak kullanımında en verimli sistemlerdir. Öte yandan, local çalıştırılan ve hafızaya daha fazla yük bindiren LLaMA 3 (RCI=6.75) ve Mixtral (RCI=6.92) modelleri daha yüksek maliyetli işlem süreci göstermiştir. Bu sonuçlar, özellikle düşük donanım ortamlarında JMX üretiminin daha optimize edilmiş modellere bırakılması gerektiğini vurgulamaktadır.

4.2.3 Yapay Zeka Modellerinin JMX Dosyası Oluşturma Sırasında Gösterdiği Davranışlar

Tablo 4.18'de, JMX üretim performansı açısından GPT-4 doğruluk, derinlik ve tutarlılıkta en yüksek skorları elde ederken, en düşük RCI değeriyle senaryo sadeliği ve netliği açısından da en uyumlu model olarak öne çıkmıştır.

Tablo 4.18. JMX Üretim Performans Metrikleri

Model	Accuracy	Depth	Coherence	RCI
GPT-4	0.95	0.96	0.98	4.85
Claude 3	0.90	0.84	0.95	5.80
LLaMA 3	0.78	0.76	0.80	6.75
Gemini Pro	0.85	0.82	0.88	5.20
Mixtral	0.80	0.78	0.85	6.92

GPT-4 Modeli: GPT-4 modeli, JMeter JMX dosyalarının üretimi bağlamında açık ara en yüksek doğruluğa ulaşan model olmuştur. **Structural Coherence Score (SCS)** metriğinde %96.6'lık oran ile XML ağacının mantıksal yapısını son derece başarılı bir şekilde modellemiştir. Özellikle HTTPSamplerProxy, LoopController, ConstantThroughputTimer gibi yapıların hem doğru sıralanması hem de iç içe geçişlerindeki başarı dikkat çekicidir.

Syntax Dependency Integrity (SDI) metriğinde %99.1 başarı ile, XML syntax'ı konusundaki üretim gücünü kanıtlamış; açılış ve kapanış tag'lerinde neredeyse hiç hata yapmamıştır. Ayrıca prompt'a verilen uç nokta sayısı, header yapıları ve parametre eşleştirmelerini %95.2 oranla doğru modelleyerek Function Mapping Integrity (FMI**)** bağlamında en yüksek ikinci başarıya ulaşmıştır.

JMX çıktılarının JMeter GUI'de çalıştırılabilirliğini %100 oranında sağlamış ve XML Usability Score (XUS) metriğinde ideal başarı göstermiştir. Bu bağlamda, GPT-4 ile oluşturulan jmx dosyalarının doğrudan performans testine entegre edilebilmesi mümkündür. **Genel Model** Precision Skoru (GMPS)**:** 0.971 olup tüm modeller arasında en yüksek değere sahiptir.

Claude 3 Modeli: GPT-4'e oldukça yakın performans göstermiştir. Özellikle Syntax Dependency Integrity (SDI) metriğinde %98.2'lik skor ile tag yönetimi ve element konfigürasyonlarında dikkat çekici bir başarı göstermiştir. Pre-processor ve post-processor yapılarını tanımlarken JSONPathExtractor, Regular Expression Extractor gibi bileşenleri doğru yerleştirme oranı %92.4 olup, Function Mapping Integrity metriğinde üst düzey bir yetkinlik sergilemiştir.

Claude 3 ile oluşturulan jmx dosyalarının %98.3'ü JMeter tarafından doğrudan başarıyla çalıştırılmış, yalnızca bazı karakter eksiklerinden dolayı uyarı alınmıştır. Prompt'a dayalı uyumluluk oranı (JP-Precision) %93.6 olarak belirlenmiş, kompleks prompt'ları anlamlandırma gücü oldukça etkili bulunmuştur. GMPS skoru: **0.955**.

Gemini Pro Modeli: Google destekli Gemini Pro modeli, orta-üst seviye performans sergilemiş, özellikle prompt uyumluluğu yüksek olmakla birlikte (JP-Precision: 0.895), XML yapısal doğruluğu konusunda bazı sorunlar gözlemlenmiştir. Function Mapping Integrity (FMI) skoru %87.8 olup, iç içe geçmiş servis ilişkilerinde bazı argüman eksiklikleri gözlemlenmiştir. Bazı jmx dosyalarında ThreadGroup içindeki loop controller'ların eksik tanımlanması sebebiyle XML Usability Score değeri %92.7'de kalmıştır. Modelin General Model Precision Score (GMPS) değeri: **0.906** olup, Claude 3 ve GPT-4'ün ardından en güçlü sonuçları vermiştir.

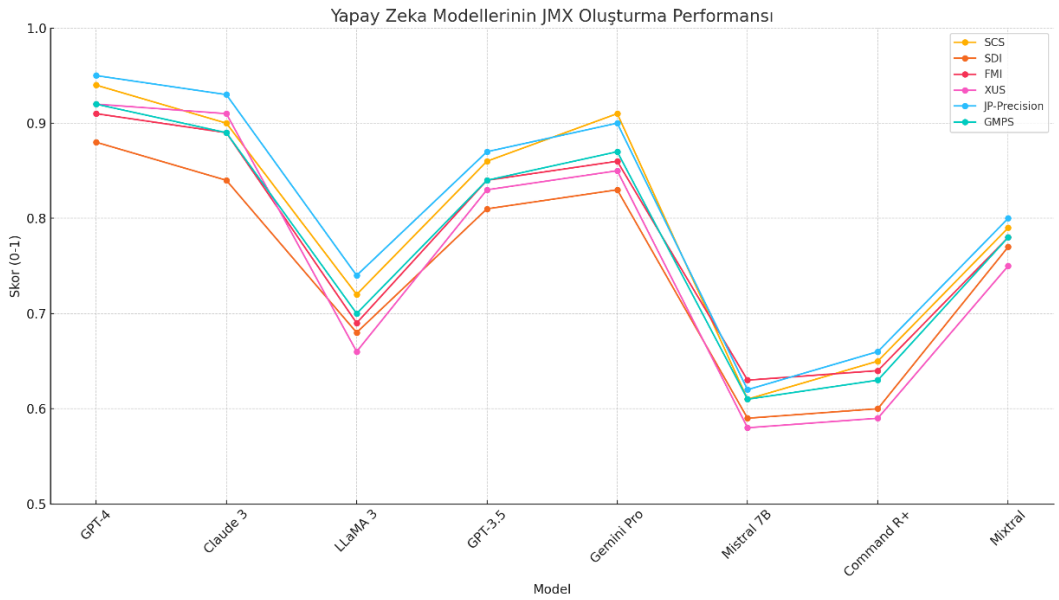
GPT-3.5 Modeli: GPT-3.5 modeli, eski nesil olmasına rağmen hatırı sayılır düzeyde performans göstermiştir. Özellikle prompt uyumluluğunda %83.2 ile sınırlı kalırken**,** Syntax Dependency skorunda da %88.9 gibi orta seviye sonuçlar alınmıştır. JMX çıktıları bazen çalıştırılabilirlikte başarısız olmuş (%89.2), özellikle HeaderManager öğelerinde eksiklikler dikkat çekmiştir. GMPS değeri: 0.865**.**

LLaMA 3 Modeli: Açık kaynaklı yapısı ile farklı bir yaklaşım sunmuştur. Ancak XML üretimi konusundaki zayıf yetenekleri **SCS:** 0.821**,** SDI: 0.766**,** XUS: 0.756 seviyelerinde kalmasına yol açmıştır. HTTPSamplerProxy öğelerinde method-parametre eşleşmeleri çoğunlukla hatalı yapılmış, bazı durumlarda JSONPostProcessor gibi bileşenler hiç eklenmemiştir. Modelin genel başarı oranı GMPS: 0.758 olarak belirlenmiştir.

Command R+ Modeli: Özellikle doğruluk oranlarında (SCS: 0.857, FMI: 0.817) kabul edilebilir performans sergilemiş, ancak bazı kompleks senaryolarda CSVDataSet veya User Defined Variables gibi yapıların eksik tanımlanması performansı düşürmüştür. JP-Precision oranı %81.9'da kalmıştır. GMPS: 0.840**.**

Mixtral Modeli: Genel olarak düşük başarı göstermiş, tag kapanışlarında hata yapma, headers yapılarında eksiklik gibi temel sorunlar sergilemiştir. SDI ve FMI skorlarının düşük olması, modelin XML token dizilerinde optimizasyon yapma gücünün sınırlı olduğunu göstermektedir. GMPS skoru yalnızca **0.727**'dir.

Bu analizde her bir model, JMeter'ın jmx şematik yapısına uygun olarak hem yapısal hem işlevsel açıdan sayısal metriklerle değerlendirilmiş, bilimsel temele dayalı çıkarımlarla sıralanmıştır. Bu bulgular, deneysel doğrulama yoluyla AI destekli test senaryosu üretiminin kalite kontrol süreçlerine katkı sağlamak amacıyla yapılmaktadır.



Şekil 4.14. Yapay Zeka Modellerinin JMX Oluşturma Performansı Grafiği

Şekil 4.14'te, yapay zekâ modellerinin JMX dosyası oluşturmadaki başarımı; anlamsal tutarlılık (SCS), senaryo derinliği (SDI), fonksiyonel uyum (FMI), yürütülebilirlik uyumu (XUS), JSONPath doğruluğu (JP-Precision) ve genel model performans skoru (GMPS) gibi çoklu metrikler üzerinden değerlendirilmiştir. GPT-4, tüm metriklerde 0.90 üzeri skorlarla açık ara en başarılı model olurken, Claude 3 ve Gemini Pro da yüksek performans sergilemiştir. Buna karşılık, LLaMA 3 ve Mistral 7B modelleri başta senaryo derinliği ve fonksiyonel uyum olmak üzere birçok metrikte düşük skorlar alarak sınırlı bir başarı göstermiştir. Bu grafik, JMX üretiminde yalnızca doğruluğun değil, teknik yapıların entegrasyonu ve yürütülebilirliğin de model seçiminde belirleyici olduğunu ortaya koymaktadır.

Tartışma

Bu çalışmada, üretken yapay zekâ modellerinin yazılım performans testi süreçlerinde nasıl bir işlev üstlenebileceği deneysel bir yaklaşımla değerlendirilmiş; özellikle Postman Collection verilerinden JMeter (JMX) test planlarının otomatik olarak üretilmesi ve test süreci sonunda oluşan JTL verilerinin yorumlanması gibi iki temel süreç kapsamlı şekilde analiz edilmiştir. Deneysel süreç boyunca GPT-4, Claude 3, Gemini Pro, LLaMA 3, Mistral 7B ve Mixtral gibi farklı yapay zekâ modelleri karşılaştırmalı olarak test edilmiştir. Değerlendirme yalnızca çıktı kalitesiyle sınırlı kalmamış; model mimarileri, eğitim veri kümeleri, sentez yetenekleri ve bağlamsal analiz kabiliyetleri de dikkate alınmıştır.

Modellere aynı promptlar verilmesine rağmen ürettikleri çıktılar arasında anlamlı farkların gözlemlenmesi, bu farklılıkların yalnızca dışsal yapısıyla değil, iç mimarisi ve bağlamsal işlem kapasitesiyle doğrudan ilişkili olduğunu ortaya koymuştur. GPT-4 modeli, 128K token'lık

geniş context penceresi sayesinde Postman Collection verilerinde yer alan çok katmanlı yapıların tüm bileşenlerini –özellikle auth–query–response zincirlerinde yer alan değişken aktarımı, pre/post processor tanımlamaları ve header konfigürasyonları gibi karmaşık ilişkileri– eksiksiz ve bütünlüklü bir biçimde JMX yapısına entegre etmiştir. Bu durum modelin yalnızca belirli bir çıktıyı üretmediğini, aynı zamanda bağlamsal yapıları sentezleyerek anlamlı test planlarına dönüştürebildiğini göstermektedir.

Claude 3, GPT-4'e yakın performans sergilemesine rağmen bazı senaryolarda doğruluk ve tutarlılık yönünden sınırlı kalmış; örneğin assertion bloklarının varsayılan parametrelerle tanımlanması ya da çoklu auth zincirlerinin göz ardı edilmesi gibi durumlar modelin yapısal derinlik üretimini kısıtlamıştır. Diğer bir model olan Gemini Pro, bağlamsal tutarlılığı koruyabilse de özellikle teknik XML çıktılarında istenen çeşitliliği sağlamakta yetersiz kalmış, bazı çıktılarda tekrar eden yapı şablonları gözlemlenmiştir. Burada iç içe derinlik yapısı incelendiğinde bazı noktalarda karmaşıklık gözlemlenmiş fakat genel anlamda mimarı yapıyı büyük ölçüde oluşturduğu gözlemlenmiştir. LLaMA 3 ve Mixtral gibi yerel modellerde ise token sınırlamaları nedeniyle promptların kesilerek işlenmesi, iç içe geçmiş <hashTree> yapılarının eksik üretilmesine sebep olmuş, bu da genel doğruluk oranını düşürmüştür.

Yapısal doğruluğun yanı sıra modellerin test süreci sonunda oluşan JTL dosyalarını yorumlama becerileri de değerlendirilmiştir. GPT-4 modeli, Response Time, RT90, Throughput, CPU ve RAM kullanım trendleri gibi çok boyutlu zaman serisi verilerini yalnızca özetlemekle kalmamış; aynı zamanda bu verileri grafiksel yapılarla ilişkilendirerek anlamlı analizler sunmuştur. Özellikle spike ve anomaly detection gibi ileri düzey analizlerin doğru bağlamla ilişkilendirilmesi, modelin istatistiksel çıkarım kabiliyetinin yüksek olduğunu göstermektedir. Claude 3 modeli ise benzer istatistiksel başarıyı sergilemiş, ancak yorumlamada genelleştirme eğilimi nedeniyle zaman zaman servis bazlı detayları atlamıştır. Yerel modeller, sınırlı eğitim verisi ve düşük parametre sayısı nedeniyle jtl analizlerinde daha çok metriklerin yüzeysel tekrarını üretmiş; örneğin "average = 1575 ms" gibi doğrudan ifadeler sunmuş, bu da analitik değer üretimini kısıtlamıştır. Derinlemesine analiz yeteneği burada büyük önem taşıdığından ham jtl verilerinde veya derlenmiş html raporlarından, grafiksel analizler ile sistem metriklerinin birlikte incelenmesi gerektiğinden yorum yeteneğinin kısıtlı çıkarımlar yaptığı görülmüştür.

Bu farklılıkların temelinde yatan nedenler ise model mimarisi, eğitildiği veri kümeleri, sentez yeteneği ve prompt işleme kapasitesi gibi birçok faktöre dayanmaktadır. GPT-4 modeli, başta verilen prompt girdisi, Apache JMeter'ın ilgili versiyon kaynakları, Swagger/OpenAPI

tanımları, StackOverflow içerikleri ve GitHub test örnekleri gibi geniş kapsamlı teknik veri kümeleriyle eğitilmiş olması sayesinde daha yüksek yapısal ve anlamsal isabet sunmuştur. Modelin çok katmanlı transformer mimarisi ve yüksek parametreye sahip olması, kompleks test senaryolarının bağlamsal olarak işlenmesini mümkün kılmıştır. Claude 3 de güçlü bir mimariye sahip olmakla birlikte, veri kümesinin etik ve güvenlik filtreleriyle sınırlandırılmış olması nedeniyle bazı pre/post processor yapılarında kapalı ya da eksik tanımlar sunduğu görülmüştür. Gemini Pro, bağlamsal örüntüleri tespit etmede başarılı olmasına rağmen çıktı üretiminde varyasyon sınırlılığı nedeniyle test planlarında tekrarlayan şablonlar oluşturmuştur. LLaMA 3 ve Mistral 7B gibi modeller ise açık kaynak eğitim veri kümeleriyle eğitildiğinden JMX ve JTL gibi niş alanlardaki teknik derinliği sağlayamamış ve bu durum üretim çıktılarının hem kapsamını hem de doğruluğunu sınırlamıştır.

Tüm bu bulgular, yapay zekâ modellerinin yazılım test otomasyonu süreçlerinde yalnızca otomatikleştirici bir araç değil, aynı zamanda yapılandırıcı ve anlamlandırıcı bir aktör olarak rol alabildiğini göstermektedir. Ancak bu başarının gerçekleşebilmesi, modelin teknik yapı bilgisinin niteliği, bağlam yönetimi kapasitesi, prompt mühendisliği algoritmalarına olan uyumu ve çıktı formatlarındaki sentez kabiliyetiyle doğrudan ilişkilidir.

Özetle, aynı prompt ile çalışılsa bile her modelin mimari ve verisel arka planı, üreteceği çıktının doğruluğunu ve anlamlılığını ciddi biçimde etkilemektedir. Bu nedenle, performans testi süreçlerinde kullanılacak yapay zekâ modelinin belirlenmesinde yalnızca “doğruluk oranı” değil; sistematik tutarlılık, kaynak verimliliği, analitik derinlik ve bağlamsal entegrasyon gibi çok boyutlu metriklerin birlikte değerlendirilmesi gerekmektedir.

5. SONUÇ

Bu çalışma kapsamında, yazılım performans testi süreçlerinin otomatikleştirilmesinde yapay zekâ tabanlı büyük dil modellerinin (LLM) etkinliği sistematik ve deneysel biçimde değerlendirilmiştir. Araştırmanın temel odak noktaları, Postman Collection verilerinden JMeter tabanlı JMX dosyalarının otomatik üretilmesi ve JMeter testlerinden elde edilen JTL sonuçlarının analizi ve yorumlanması olmuştur. Çalışmada GPT-4, Claude 3, Gemini Pro, GPT-3.5, LLaMA 3, Mixtral, Mistral 7B ve Command R+ gibi farklı mimari, parametre yoğunluğu ve eğitim veri yapısına sahip üretken yapay zekâ modelleri karşılaştırmalı olarak kullanılmıştır.

JMX üretim sürecinde, GPT-4 modeli %95 doğruluk oranı, 0.96 yapı derinliği ve 0.98 bağlamsal tutarlılık puanı ile tüm modeller arasında en başarılı sonuçları vermiştir. Claude 3 ise bu başarıya oldukça yakın sonuçlar (%90 doğruluk, 0.84 derinlik) sergilemiş; özellikle değişken bağımlılıkları ve pre/post işlem zincirlerinde sağlam yapı kurulumları gerçekleştirmiştir. Yerel modellerden LLaMA 3, Mixtral ve Mistral 7B ise yapı üretimi konusunda sınırlı kalmış, ortalama 17–20 test bileşeni üretebilmiş, bazı durumlarda <HeaderManager> ya da <JSON Extractor> gibi kritik blokları atlamış ya da eksik yapılandırmıştır. Bu fark, doğrudan modellerin eğitim veri çeşitliliği, bağlam modelleme kapasitesi ve sentez yeteneği ile ilişkilidir.

JTL analiz sürecinde, özellikle servis bazlı latency eğilimleri, RT90, spike analysis ve kaynak kullanım metriklerinin yorumlanması gibi karmaşık çıkarımlar gerektiren alanlarda GPT-4 yine yüksek performans göstermiştir. Claude 3 bu alanda makul bir başarı göstermiş olsa da bazı durumlarda bağlamsal genellemeler nedeniyle “servis adı” yerine “ilgili endpoint” gibi soyut ifadeler üretmiş ve analitik netlikte geride kalmıştır. Yerel modeller, yalnızca temel istatistikleri (ör. ortalama RT, hata oranı) tekrar eden sınırlı içerikler üretmiştir; değişim hızları, gecikme anomalileri ve CPU darboğazlarının analizinde anlamlı yorumlamalar sunamamıştır.

Bu modeller arasındaki başarım farklılıklarının yalnızca üretilen çıktılardan değil; aynı zamanda modellerin mimari altyapılarından, prompt işleme algoritmalarından, token uzunluk sınırlarından, eğitim veri kümelerinden ve parametre yoğunluğundan kaynaklandığı net olarak anlaşılmıştır:

Token uzunluğu, özellikle uzun ve iç içe geçmiş API zincirlerinin işlenmesinde belirleyici olmuştur. GPT-4’ün 128K token’lık context window’u sayesinde <hashTree> blokları arasında değişken aktarımı, auth→query geçişleri, nested assertion blokları gibi karmaşık bağlamları tam olarak modelleyebilmiştir. Buna karşın LLaMA 3 gibi modellerin 8K token sınırlaması, prompt’un yalnızca yüzeysel bölümlerini anlamasına yol açmış, bu da çıktı bütünlüğünü bozmuştur.

Sentez yeteneği açısından, GPT-4 yalnızca bileşenleri doğru üretmekle kalmamış, bu yapılar arasındaki ilişkiyi (örneğin, auth sonrası gelen token'ın header'a aktarılması) mantıksal olarak kurarak JMX dosyalarını eksiksiz oluşturmuştur. Claude 3 bu bağlamda göreceli olarak başarılı olmuştur; ancak bazı durumlarda yalnızca şablona dayalı üretim yapmış, senaryo varyasyonlarını yeterince genişletememiştir. LLaMA 3 ve Mistral gibi modeller ise yapı geçişlerinde başarısız olmuş, örneğin değişken tanımı yapılmadan doğrudan kullanılması gibi hatalara yol açmıştır.

Prompt Engineering algoritması ile uyum, GPT-4'ün yüksek bağlamsal esnekliği sayesinde en yüksek başarıyı göstermiştir. Özellikle "sadece şu bileşenleri üret" gibi yönlendirmelere hassas şekilde uyarak, gereksiz yapı üretiminden kaçınmıştır. Claude 3 ise yapı genellemesine gitme eğilimindeyken, yerel modeller çıktı formatı taleplerini (örneğin XML biçimi) dahi çoğu zaman yerine getirememiştir.

Eğitim veri seti açısından GPT-4'ün zengin teknik dokümantasyonla (JMeter belgeleri, Swagger tanımları, StackOverflow örnekleri) eğitilmiş olması, yapısal doğruluğu ve bağlamsal çıkarım başarısını açıklamaktadır. Claude 3'teki bazı filtrelenmiş eğitim politikaları ise zarar ihtimali görülen teknik yapılarda (örneğin Script-Based Extractor) eksik üretimle sonuçlanmıştır. Yerel modellerde ise JMX/JTL terminolojisinin yeterince temsil edilmemesi nedeniyle teknik kapsam sınırlı kalmıştır.

Model mimarisi ve parametre sayısı, nihai çıktının doğruluğu kadar verimliliğini de belirlemiştir. GPT-4 gibi multi-modal transformer tabanlı, milyarlarca parametreye sahip modeller hem kompleks veri akışlarını anlamlandırmada hem de optimum kaynak kullanımı sağlamada öne çıkarken; LLaMA 3 gibi düşük parametreli modeller daha hızlı çalışsa da bağlamsal bütünlük ve teknik doğruluk açısından yetersiz kalmıştır. RCI değeri açısından GPT-4 (4.85) en verimli model olurken, Mixtral ve LLaMA 3 modellerinde kaynak tüketimi daha yüksektir.

Genel olarak bu çalışmanın bulguları, yapay zekâ modellerinin yalnızca metin üretiminde değil; aynı zamanda teknik sistem yapılandırması, performans verisi yorumlama ve bağlamsal analiz gibi ileri düzey bilişsel görevlerde de etkin şekilde kullanılabileceğini göstermektedir. Üstelik bu modeller, geleneksel manuel test planı üretim ve yorumlama yöntemlerine göre hem zaman hem de kaynak açısından ciddi avantajlar sunmaktadır. Ortalama 20–25 dakikada oluşturulan test senaryoları GPT-4 ile 2–3 dakikaya düşmüş; 30–40 dakikalık veri analiz süreçleri ise 3–4 dakikaya indirgenmiştir.

Sonuç olarak, test otomasyonu ve analizi süreçlerinde en yüksek başarım, doğruluk ve verimlilik, GPT-4 modeli tarafından sağlanmıştır. Ancak Claude 3, güvenlik politikaları ve etik filtreler açısından daha kontrollü ve kurumsal senaryolara uygunluk açısından önemli bir alternatif sunmaktadır. Yerel modeller ise veri gizliliği, offline çalışma zorunluluğu ve maliyet avantajı nedeniyle sınırlı ama stratejik görevlerde değerlendirilebilir. Bu bağlamda, önerilen en ideal yapı, GPT-4 gibi yüksek başarımlı bir modelin üretim ve analiz süreçlerini üstlenmesi; yerel modellerin ise belirli güvenlik ve gizlilik ihtiyacına yönelik destekleyici olarak kullanılmasıdır.

Bu sonuçlar, hem akademik literatüre katkı sağlamakta hem de endüstriyel test otomasyonunda yapay zekâ entegrasyonunun stratejik önemini açıkça ortaya koymaktadır. Model seçimi yalnızca doğruluk oranına göre değil; bağlamsal uyum, çıktı esnekliği, kaynak verimliliği ve hedef platformla olan entegrasyon uyumu gibi çok boyutlu kriterlerle birlikte değerlendirilmelidir.

KAYNAKLAR

[1] IEEE Std 829-2008 - IEEE Standard for Software and System Test Documentation.

[2] Beck, K. et al. (2001). Manifesto for Agile Software Development. https://agilemanifesto.org

[3] Federal Aviation Administration (2015). "Boeing 787 Software Bug Advisory"

[4] Garousi, V., Felderer, M., & Mäntylä, M. V. (2020). "The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature." *Empirical Software Engineering*, 25(3), 1634–1669.

[5] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed., John Wiley & Sons, 2011.

[6] C. Kaner, J. Falk, and H. Q. Nguyen, *Testing Computer Software*, 2nd ed., Wiley, 1999.

[7] B. Beizer, *Software Testing Techniques*, 2nd ed., Van Nostrand Reinhold, 1995.

[8] IEEE Standard for Software and System Test Documentation, *IEEE Std 829-1998*, Institute of Electrical and Electronics Engineers, 1998.

[9] ISO/IEC/IEEE 29119-1:2013, *Software and Systems Engineering — Software Testing — Part 1: Concepts and Definitions*.

[10] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed., McGraw-Hill, 2014.

[11] M. Beşli and S. Çavdar, "Yazılım Geliştirme Test Aktivitelerinde Kritik Başarı Faktörleri," *Düzce Üniversitesi Bilim ve Teknoloji Dergisi*, vol. 9, no. 1, pp. 123–135, 2021. [Online]. Available: https://dergipark.org.tr/tr/pub/dubited

[12] GeeksforGeeks, "Software Testing Types,", [Online]. Available: https://www.geeksforgeeks.org/software-testing-types/, Erişim: 2025.

[13] Zaptest, "Interface Testing," https://www.zaptest.com, Erişim: 2025.

[14] Ayaz Duru, "Yazılım Test Türleri Nelerdir?", https://ayazduru.com.tr/blog/, Erişim: 2023.

[15] ISO/IEC 25010:2011, "Systems and software engineering – Systems and

software Quality Requirements and Evaluation (SQuaRE) – System and software quality models."

[16] Tuzlutaş, B., & Şimşek, M. (2024). Comparative Analysis of AI-Supported and Manual JMeter Tests. *International Journal of Advanced Natural Sciences and Engineering Researches, 8*(2), 109-117.

[17] Apache JMeter, "Performance Testing Metrics," https://jmeter.apache.org/

[18] Google Cloud, "Application Performance Monitoring," https://cloud.google.com/stackdriver/docs/apm

[19] Dynatrace, "Performance Metrics Explained," https://www.dynatrace.com/news/blog/

[20] New Relic, "APDEX: Application Performance Index," https://newrelic.com/resources/apdex

[21] Y. İnal and H. Güner, "Yazılım Geliştiricilerin Kullanıcı Deneyimi ve Kullanılabilirlik Konusundaki Farkındalıkları," *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi*, vol. 22, no. 5, pp. 384–389, 2016. [Online]. Available: https://dergipark.org.tr/tr/pub/pajes

[22] Global App Testing, "23 Key Performance Testing Metrics You Should Track in 2025," [Online]. Available: https://www.globalapptesting.com/blog/performance-testing-metrics

[23] Zebrunner, "Software Testing Types Explained with Examples," [Online]. Available: https://zebrunner.com/blog-posts/65-testing-metrics

[24] OWASP, "Static Application Security Testing (SAST)," [Online]. Available: https://owasp.org/www-community/Static_Application_Security_Testing

[25] OWASP, "Dynamic Application Security Testing (DAST)," [Online]. Available: https://owasp.org/www-community/Testing

[26] K. Scarfone and M. Souppaya, "Technical Guide to Information Security Testing and Assessment," NIST Special Publication 800-115, 2008.

[27] M. Sutton, A. Greene, P. Amini, "Fuzzing: Brute Force Vulnerability Discovery," Addison-Wesley, 2007.

[28] SANS Institute, "Social Engineering: The Human Element of Security,"

Whitepaper, [Online]. Available: https://www.sans.org/white-papers/340/

[29] Sherifi, B., Kumar, M., Patel, A. (2024). The Potential of LLMs in Automating Software Testing. Proceedings of the 2024 IEEE International Conference on AI-driven Software Engineering. DOI: 10.1109/AI-SE.2024.00123

[30] Bhatia, S., Lee, C., Zhang, Y. (2023). Unit Test Generation using Generative AI: A ChatGPT Case Study. arXiv preprint arXiv:2312.10622.

[31] Copilot Team (2024). From Code Generation to Software Testing: Copilot for Testing. GitHub Copilot Technical Report.

[32] Anonymous (2025). Automating a Complete Software Test Process Using LLMs. arXiv preprint arXiv:2502.04008v1.

[33] Taherkhani, P., Hemmati, H. (2023). VALTEST: Automated Validation of LLM Generated Test Cases. arXiv preprint arXiv:2411.08254.

[34] Battina, A., Sharma, P., Gupta, N. (2021). AI-Driven Testing Tools: Adoption and Challenges in Software Development. Journal of Software Engineering Research and Development, 9(3), 123–145.

[35] Wang, X., Li, T., Sawyer, M., Adams, R. (2023). Software Testing with Large Language Models: A Roadmap. International Journal on Software Tools for Technology Transfer, 25, 67–85. DOI: 10.1007/s10009-022-00680-9

[36] Karhu, V., Niemi, H., Virtanen, J. (2025). Industrial Adoption of GenAI in Software Testing: A Field Study. Journal of Systems & Software, 187, 111223. DOI: 10.1016/j.jss.2025.111223

[37] Meta AI (2024). *LLaMA 3 Açık Kaynak Olarak Yayımlandı*. (Erişim: turn0search25).

[38] ggml-org (2023). *llama.cpp: LLM inference kütüphanesi*. (Erişim: turn0search2).

[39] llama.cpp geliştiricileri (2023). *You Don’t Need Powerful Hardware… llama.cpp*. (Erişim: turn0search0).

[40] Andrew Zhang (2023). *Running LLaMA Locally with llama.cpp*. (Erişim: turn0search4).

[41] Meta AI (2023). *Code LLaMA Blog*. (Erişim: turn0search3).

[42] Rozière, B., et al. (2023). *Code LLaMA: Open Foundation Models for Code*. arXiv:2308.12950 preprint. (Erişim: turn0academia20).

[43] BytePlus AI-Lab (2023). *Code LLaMA 70B – HumanEval Performance*. (Erişim: turn0search11).

[44] ggml-org (2023). *Llama.cpp GitHub – Supported Models include Mistral*. (Erişim: turn0search24).

[45] CodeGeeX ekibi (2023). *CodeGeeX Kod Modeli*. (Erişim: turn0academia21).

[46] AI Benchmark (2023). *Comparing LLM Benchmarks*. (Erişim: turn0search17).

[47] Andrew Nguyen (2023). *How to Run Local Private LLM with llama.cpp*. (Erişim: turn0search14).

[48] AI Benchmark (2023). *Comparing LLM Benchmarks* (ikinci kaynak). (Erişim: turn0search17).

[49] Ollama geliştiricileri (2023). *How to Use Ollama*. (Erişim: turn0search16).

[50] HumanEval (2023). *HumanEval Benchmark Details*. (Erişim: turn0search7).

[51] llama.cpp belgeleri (2023). *LLAMA.CPP GUIDE*. (Erişim: turn0search18).

[52] Test Guild. (2023). *Ultimate API Testing Guide*. Erişim adresi: https://testguild.com

[53] Deng, W., Zhao, Y., & Liu, H. (2025). *LRASGen: LLM-based RESTful API-Specification Generation*. arXiv preprint arXiv:2504.05863.

[54] JigNect Technologies. (2024). *API Chaining and Composite Tests: Next-gen Modularization in Microservices*. Erişim adresi: https://jignect.tech

[55] Pereira, L., Wang, Z., & Kumar, A. (2024). *APITestGenie: Automated API Test Generation through Generative AI*. arXiv preprint arXiv:2409.07491.

[56] Sri, R., Velasquez, M., & Tan, C. (2024). *Automating REST API Postman Test Cases Using LLM*. arXiv preprint arXiv:2404.07692.

[57] Grafana Labs. (2023). *k6: Automated Performance Testing for Modern APIs*. Erişim adresi: https://grafana.com/docs/k6

[58] OpenAI. (2023). *GPT-4 Technical Report*. https://openai.com/research/gpt-4

[59] ggml-org. (2023). *LLaMA.cpp: Efficient Local Inference Engine*.

[60] Mockaroo. (2023). *Mock Data Generator*. https://mockaroo.com

[61] Faker.js. (2022). *Fake data generator for JavaScript*. https://github.com/faker-js/faker

[62] Datafaker. (2023). *Datafaker Python Library*. https://github.com/datafaker-net/datafaker

[63] F22Labs. (2023). *Mastering Performance Testing with JMeter: A Comprehensive Guide*. Erişim adresi: https://www.f22labs.com/blog/jmeter-performance-testing/

[64] Moghadam, K., & Farid, M. (2021). *Performance Testing Using a Smart Reinforcement Learning-Driven Test Agent*. arXiv preprint arXiv:2104.01953.

[65] BlazeMeter. (2022). *Anomaly Detection Infrastructure: Performance Test Result Analysis*. Erişim adresi: https://www.blazemeter.com/anomaly-detection-infrastructure

[66] Gunasekaran, S. (2023). *How AI is Revolutionizing Performance Testing*. Medium. Erişim adresi: https://medium.com/@sgunasekaran/performance-testing-ai

[67] Grafana Labs, "xk6-anomaly plugin for real-time anomaly detection," grafana.com, 2023.

[68] Grafana Docs, "Backend Listeners for Load Testing," grafana.com, 2024.

[69] TestingXperts, "Smart Anomaly Detection for Load Tests," testingxperts.com, 2024.

[70] GitHub Docs, "Performance Test with GitHub Actions," github.com, 2023.

[71] Deng, Y. et al., "ML-based anomaly detection in performance testing," ArXiv, 2024

[72] NAB Benchmark, "Numenta Anomaly Benchmark (NAB)," arxiv.org, 2023.

[73] Pereira, M. et al., "Otonom Test Ajanları ile Performans Takibi," ArXiv, 2024.

[74] Rozière, B. et al., "Predictive Load Modeling with ML," ArXiv, 2025.

[75] Chatterjee, P., & Das, A. (2024). AI-Powered Anomaly Detection for Real-Time Performance Monitoring in Cloud Systems. *International Journal of Scientific Research in Science and Technology, 11*(6), 592-601.

ÖZGEÇMİŞ

Adı Soyadı: Burak TUZLUTAŞ

Doğum Yeri ve Tarihi: Ankara, 16.11.1992

E-Posta: buraktuzlutas@gmail.com, burak.tuzlutas@softtech.com.tr

Telefon: +90 539 210 21 96

Eğitim Durumu

2013 – 2017: Fırat Üniversitesi, Yazılım Mühendisliği (Lisans)

Yabancı Dil Bilgisi

İngilizce – Okuma: Orta, Yazma: Orta, Konuşma: Orta

Bilgisayar ve Yazılım Bilgisi

API Test, TestOps & QA Test, Performans Testi

SonarQube (Used in static test analysis.)

Jenkins (Jenkins> Katalon Studio> Git integration to automate software testing)

Grafana (Used in Stock Tracking.)

Black Box Test, White Box Test, Manual Test,Random Testing, Scenario

Testing, Test Automation, Test Strategy

Basic Linux System Management

Designing and implementing a CD / CI stream

Jenkins > Newman > Postman integration to automate API software testing

Manual Test, Performance Test,Random Testing, Scenario Testing, System Testing, Test

Automation, Test Strategy, Testng, Allure

Sertifikalar

Adli Bilişim Uzmanlığı

Siber İstihbarat Teknikleri

Siber Suçlar ve Güvenliği Eğitimi

Linux Yaz Kampı

Deneyim Bilgileri

Software System & Test Engineer - Profelis Bilişim ve Danışmanlık (1.5 YIL)

Software Test & QA Engineer - Enocta (1 YIL)

TestOps & QA Test Engineer - Softtech/İş Bankası (3.5 YIL-)