MAKİNE ÖĞRENMESİ DESTEKLİ SCRUM PLANLAMASI OTOMASYONU

Yekta Buğrahan NİZAMOĞLU
221402101

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı
Danışman: Dr. Öğr. Üyesi Önder ŞAHİNASLAN

İstanbul
T.C. Maltepe Üniversitesi
Lisansüstü Eğitim Enstitüsü
Ağustos, 2024

MAKİNE ÖĞRENMESİ DESTEKLİ SCRUM PLANLAMASI OTOMASYONU

Yekta Buğrahan NİZAMOĞLU
221402101
ORCID:0009-0001-2162-0423

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı
Danışman: Dr. Öğr. Üyesi Önder ŞAHİNASLAN

İstanbul
T.C. Maltepe Üniversitesi
Lisansüstü Eğitim Enstitüsü
Ağustos, 2024



ETİK İLKE VE KURALLARA UYUM BEYANI

YEKTA BUĞRAHAN NİZAMOĞLU'ın "Makine Öğrenmesi Destekli Scrum Planlama Otomasyonu" başlıklı tezi 22.08.2024 tarihinde aşağıdaki jüri tarafından değerlendirilerek "Maltepe Üniversitesi Lisansüstü Eğitim ve Öğretim Yönetmeliği" nin ilgili maddeleri uyarınca Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans/Doktora/Sanatta Yeterlik tezi oy birliğiyle/oy çokluğuyla, başarılı/başarısız olarak kabul edilmiştir.

	Unvanı. Adı ve Soyadı	İmza
Üye (Tez Danışmanı)	Dr. Öğr. Üyesi Önder ŞAHİNASLAN (Danışman) Maltepe Üniversitesi	.....
Üye	Dr. Öğr. Üyesi Can RAZBONYALI Galata Üniversitesi	.....
Üye	Dr. Öğr. Üyesi Cem CANTEKİN Maltepe Üniversitesi	.....

Prof. Dr. Selva ERSÖZ KARAKULAKOĞLU

Enstitü Müdürü

TEŞEKKÜR

Araştırma, ders ve mezuniyet projesi aşamalarında her türlü yardım ve yönlendirmesinden dolayı danışmanım Dr. Öğr. Üyesi Önder Şahinaslan'a, her zaman yanımda olan ve desteğini hiçbir zaman esirgemeyen eşim Elif Duygu Nizamoğlu'na, ihtiyaç duyduğum her an motivasyon kaynağım olan kızım Asya Nizamoğlu'na teşekkürü bir borç bilirim.

Yekta Buğrahan NIZAMOĞLU

Ağustos, 2024

ÖZET

MAKİNE ÖĞRENMESİ DESTEKLİ SCRUM PLANNING OTOMASYONU

Yekta Buğrahan Nizamoğlu
Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı
Danışman: Dr. Öğr. Üyesi Önder Şahinaslan
Maltepe Üniversitesi Lisansüstü Eğitim Enstitüsü, 2024

Modern dünyada yazılım geliştirme süreçleri ele alındığında görülebilir ki, dünyanın her yerinde birbirinden farklı yazılım geliştirme ve planlama metotları uygulanmaktadır. Bunlardan en yayın olanlarından biri sürekli geliştirme ve sürekli çalışan bir ürün elde etmeyi hedefleyen, uzun planlamalar yerine kısa süreli ve ölçülebilir planlamaları tercih eden, takım bazlı daha ufak ekipler halinde çalışıp bir arada bir proje geliştirmeye imkan sağlayan, Agile tabanlı Scrum yöntemidir.

Bu yöntemde yapılacak olan işin tamamı göz önüne alınarak daha küçük parçalara bölme ve bu küçük parçalar ile adım adım, her seferinde çalışan bir ürün elde etme odaklı bir süreç planlaması söz konusudur. Bu ufak parçalar Scrum'ın bir etkinliği olan "planlama" sırasında takım tarafından puanlanıp, bu puanlanan işler sonucunda bir Sprint süresinde ne kadar iş yapılabildiğinin ölçümü ve bir sonraki sprintte ne kadar iş yapılabileceğinin öngörüsü sağlanmaktadır. Bu puanlama sırasında verilen puanlara Story Point denmektedir. Bu süreç her ne kadar proje planlaması açısından çok verimli olsa da her sprint planlamasında tüm takımın tek tek yapılacak işlerin üzerinden geçilmesini ve puanlamasını gerektirdiği için zaman açısından çok maliyetli olmaktadır. Bu çalışma kapsamında, yazılım geliştirme sürecinde takımın üzerinde özellikle zaman açısından yüksek maliyet yaratan bu puanlama sistemini, makine öğrenmesi destekli bir otomasyon sistemi ile hafifletmeyi ve daha stabil bir süreç haline getirmek amaçlanmaktadır. Bu otomasyon sistemi kurulurken genel yazılım geliştirme ve planlama süreçleri bilgisi ile makine öğrenmesi teknolojilerinden faydalanılmıştır.

Anahtar Sözcükler: Agile, Scrum, Makine Öğrenmesi, Planlama, Story Point

ABSTRACT

MACHINE LEARNIN SUPPORTED AUTOMATION IN SCRUM PLANNING

Yekta Buğrahan Nizamoğlu
Master Thesis
Department of Computer Engineering
Computer Engineering Master's Program with Thesis
Advisor: Asst. Prof. Önder Şahinaslan
Maltepe University Graduate Education Institute, 2024

When it comes to software development processes in the modern world, it can be seen that various software development and planning methods are implemented across the globe. One of the most popular ones is the Agile-based Scrum method, which aims to achieve continuous development and continuously working product delivery, preferring short-term and measurable planning over long-term planning. It allows teams to work together in smaller teams, enabling the development of a project collectively.

In this method, the entire task at hand is divided into smaller pieces, and a process plan is focused on obtaining a working product step by step with these smaller pieces. These small pieces are scored by the team during the “planning” phase, which is an activity of Scrum. By scoring these tasks, the measurement of how much work can be done in a Sprint period and the prediction of how much work can be done in the next sprint are provided. This scoring is referred as Story Points. Although this process is very efficient in terms of project planning, it becomes time-consuming because it requires the entire team to go through and score each individual task during each sprint planning.

Within the scope of this study, the aim is to mitigate the high time cost of this scoring system, which particularly burdens the team in terms of time, and transform it into a more stable process using a machine learning-supported automation system. When establishing this automation system, both general software development a planning process knowledge and machine learning Technologies have been utilized.

Keywords: Agile, Scrum, Machine Learning, Planning, Story Points

İÇİNDEKİLER

ETİK İLKE VE KURALLARA UYUM BEYANI ii
TEŞEKKÜR iii
ÖZET iv
ABSTRACT v
İÇİNDEKİLER vi
ŞEKİLLER LİSTESİ xi
TABLOLAR LİSTESİ xiii
KISALTMALAR xiv
1. GİRİŞ 1
1.1 Problem 5
1.2 Amaç 6
1.3 Çalışmanın Önemi 6
1.4 Varsayımlar 7
1.5 Limitler 7
1.6 Tanımlar 7
2. LİTERATÜR ARAŞTIRMASI 9
2.1 Makine Öğrenmesi ile Metin Kategorizasyonu 9
2.2 Büyük Dil Modelleri Üzerine Yapılan Çalışmalar 10
2.3 Scrum Süreçleri Üzerine Yapılan Çalışmalar 11
2.4 Yapılacak İşlerin Puanlanması Üzerine Yapılan Çalışmalar 11
3. GENEL BİLGİLER 13
3.1 Agile 13
3.1.1 Agile manifestonun temel değerleri 14

3.1.2 Agile manifestonun prensipleri 15
3.1.2.1 Müşteriyi tatmin et 15
3.1.2.2 Değişen gereksinimlere açık olmak 16
3.1.2.3 Müşteriye sık sık çalışan yazılım sunulması 16
3.1.2.4 Günlük olarak iş birliği sağlanması 17
3.1.2.5 Motive edilmiş bireyler 17
3.1.2.6 Yüz yüze görüşmeler 17
3.1.2.7 İlerlemenin miktarının çalışan ürünler aracılığıyla ölçülmesi 18
3.1.2.8 Sürdürülebilir gelişimin teşvik edilmesi 18
3.1.2.9 Teknik mükemmelliğe sürekli dikkat 19
3.1.2.10 Basitlik esastır 19
3.1.2.11 Kendi organize olan ekipler 19
3.1.2.12 Düzenli olarak sürekli iyileşmeyi yansıtmak 20
3.1.3 Agile proje yönetiminin avantajları ve dezavantajları 21
3.1.3.1 Agile metotlarının avantajları 21
3.1.3.2 Agile metotlarının dezavantajları 22
3.1.4 Agile metodolojilerin sık uygulanan türleri 23
3.2 Scrum 23
3.2.1 Scrum tarihçesi 24
3.2.2 Scrum teorisi 24
3.2.3 Scrum rolleri 25

3.2.4 Scrum etkinlikleri 29
3.2.4.1 Sprint 29
3.2.4.2 Sprint planlaması 30
3.2.4.3 Günlük toplantı 30
3.2.4.4 Sprint review 31
3.2.4.5 Sprint retrospective 31
3.2.5 Scrum araçları 32
3.2.5.1 Product backlog 32
3.2.5.2 Sprint backlog 33
3.2.5.3 Bitmiş iş tanımı 34
3.2.5.4 Increment 34
3.3 Makine Öğrenmesi 34
3.3.1 Makine öğrenmesi stratejileri 36
3.3.1.1 Denetimli öğrenme 37
3.3.1.2 Yarı denetimli öğrenme 38
3.3.1.3 Denetimsiz öğrenme 38
3.3.1.4 Çevrimiçi öğrenme 38
3.3.1.5 Takviyeli öğrenme 38
3.3.1.6 Aktif öğrenme 39
4. YÖNTEM 40
4.1 Kullanılan Dil Modelleri 40

4.1.1 Meta LLama dil modeli 41
4.1.1.1 Matematiksel yapı 41
4.1.1.2 Özellikler 42
4.1.1.3 Kullanım alanları 42
4.1.2 Phi 3 dil modeli 43
4.1.2.1 Matematiksel yapı 43
4.1.2.2 Özellikler 43
4.1.2.3 Kullanım alanları 44
4.2 LM Studio 44
4.2.1 LM Studio'nun işlevleri 44
4.2.2 LM Studio kullanımı 45
4.2.3 Uygulamanın ayrıntılı tanıtımı 46
4.3 Çalışmanın Yapılışı 52
4.3.1 Veri setinin oluşturulması 53
4.3.2 Veri setinin temizlenmesi 53
4.3.3 Dil modelinin eğitilmesi 55
4.3.4 Eğitilmiş olan dil modelinin testi 59
5. BULGULAR VE TARTIŞMA 63
5.1 Bulgular 63
5.1.1 Meta LLama 65
5.1.2 Phi 3 69

5.2 Tartışma 74
6. SONUÇ ve ÖNERİLER 76
7. KAYNAKÇA 77
ÖZ GEÇMİŞ 83

ŞEKİLLER LİSTESİ

Şekil 1. Waterfall Metodolijisi 2

Şekil 2. Çevik yazılım geliştirme metodolojisi 4

Şekil 3. Agile'ın değerleri 14

Şekil 4. Agile'ın prensipleri 15

Şekil 5. Scrum Ekibinde Roller 26

Şekil 6. LM Studio Giriş Ekranı 47

Şekil 7. LM Studio Arama Motoru Ekranı 47

Şekil 8. LM Studio Chat Bot Ekranı 48

Şekil 9. LM Studio Playground Ekranı 49

Şekil 10. LM Studio Local Server Ekranı 50

Şekil 11. LM Studio Modeller Ekranı 51

Şekil 12. Tez Çalışmasının Akış Diyagramı 52

Şekil 13. Veri setinin temizlenmesi ile ilgili kod 54

Şekil 14. Dil modelinin eğitilmesinde kullanılan kod 56

Şekil 15. Eğitilmiş dil modelinin testi ile ilgili kod 60

Şekil 16. Yazılım geliştirme uzmanlarının tahminleri 64

Şekil 17. Meta LLama puan tahminleri dağılımı 65

Şekil 18. Meta LLama puan tahmin sonuçları 66

Şekil 19. Meta LLama puan tahminleri kabul kriteri dağılımı 67

Şekil 20. Meta LLama kabul kriterli puan tahmin sonuçları 68

Şekil 21. Meta LLama sprint bazında hesaplanan iş hacmi 69

Şekil 22. Phi 3 puan tahminleri dağılımı 70

Şekil 23. Phi 3 puan tahmin sonuçları 71

Şekil 24. Phi 3 puan tahminleri kabul kriteri dağılımı 72

Şekil 25. Phi 3 kabul kriterli puan tahmin sonuçları 73

Şekil 26. Phi 3 sprint bazında hesaplanan iş hacmi 73

TABLOLAR LİSTESİ

Tablo 1. Meta LLama dil modeli test sonuçları 66

Tablo 2. Phi 3 dil modeli test sonuçları..... 71

KISALTMALAR

ML **:** Machine Learning

AI **:** Artificial Intelligence

SPE **:** Story Point Estimation

VSC **:** Visual Studio Code

SM **:** Scrum Master

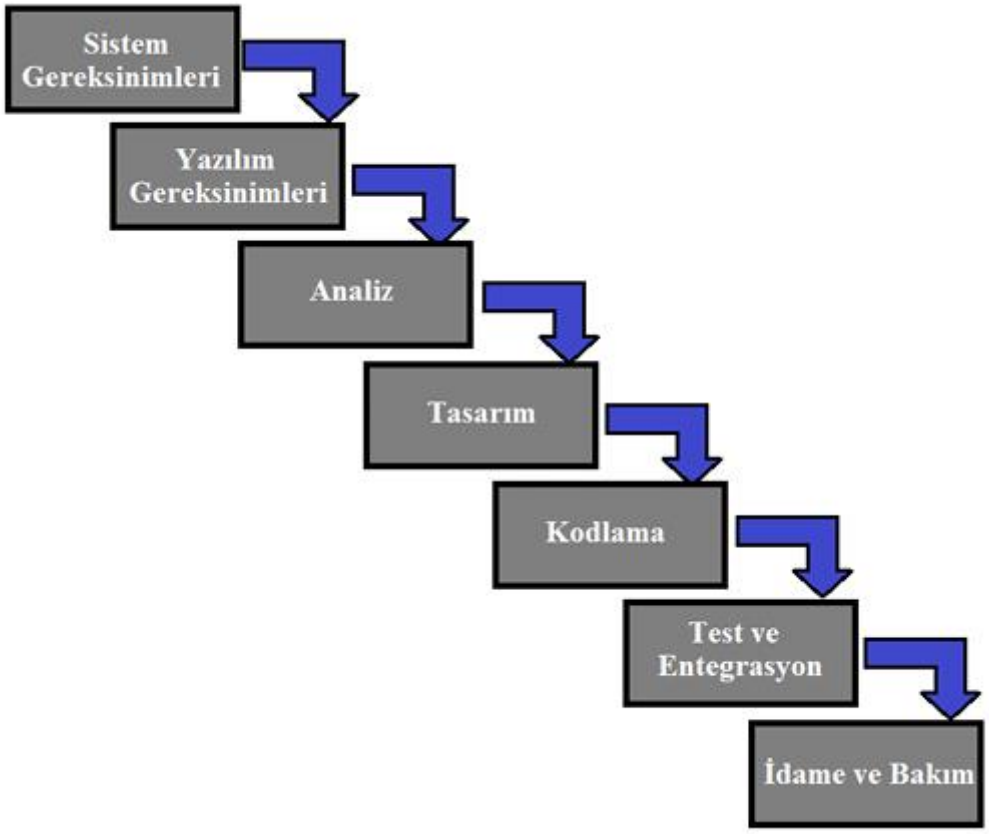
PO **:** Product Owner

1. GİRİŞ

Günümüz dünyasında, teknoloji inkâr edilemez bir şekilde günlük hayatımızın vazgeçilmez bir parçası haline gelmiştir. Günden güne telefon, otobüs, internet, sağlık kuruluşlarında kullanılan cihazlar gibi birçok alanda hayatımıza girmiştir. Tüm bu bahsi geçen cihazlar, aslında iki ana parçadan oluşmaktadır: "Donanım" ve "Yazılım". Bu iki parça birbirine o kadar bağlıdır ki, günümüz teknolojisinde biri olmadan diğeri düşünülemez.

Anlaşılacağı üzere telefon, araba, buzdolabı gibi her gün kullandığımız cihazlar ne kadar kullanışlı olurlarsa olsunlar, en verimli hallerine ancak yazılım ile desteklendiklerinde ulaşabilirler. Yazılım, donanıma maksimumu başarması için esneklik sağlamaktadır. Yazılım geliştirme süreçlerine bakacak olursak eğer, başarılı bir yazılım geliştirme sürecinin ancak bir grup profesyonelin bir arada çalışması ile mümkün olabilir (Suryanarayana, 2015). Yazılım geliştirme süreçlerinde en yaygın kullanılan metotlar, Waterfall, prototip oluşturma, yinelemeli, deneysel, sarmal geliştirme, hızlı uygulama geliştirme, ekstrem programlama ve Agile olarak sıralanabilir.

Waterfall metodolojisi uzun yıllar boyunca yazılım geliştirme dünyasında en popüler olarak kabul edilen ve kullanılan metodolojiydi. Waterfall'a göre; yazılım geliştirme süreci, isterlerin tanımlanması, analiz, tasarım, kodlama, test, kabul ve bakım gibi parçalara bölünmektedir. Bu metodun tanımlamalarına göre bir aşama tamamen bitmeden bir sonraki aşamaya başlamak mümkün değildir (McCormick, 2012). Yazılım geliştirme süreci adeta bir şelale gibi ilerler ve ismini de buradan almaktadır. Waterfall uygulanan bir yazılım geliştirme süreci Şekil 1'de olduğu gibi gösterilebilir.



Şekil 1. Waterfall Metodolijisi

Waterfall yazılım geliştirme metodolojisinde, eğer bir aşama başarısız olursa, geri kalan aşamalar da başarısız olmaktadır. Daha önce de bahsedildiği üzere, Waterfall yazılım geliştirme süreçlerinde her aşamayı tek tek takip eder. Bu davranış biçimi yazılım geliştirme süreçlerinde esnekliği ortadan kaldırır ve son dakikada bir değişiklik yapılmasına müsaade etmez. Aynı zamanda yazılım geliştirme işlemi başladığı andan itibaren yine bir değişiklik yapmak mümkün değildir çünkü tüm isterler bir önceki aşamalarda tamamen tanımlanmış haldedir.

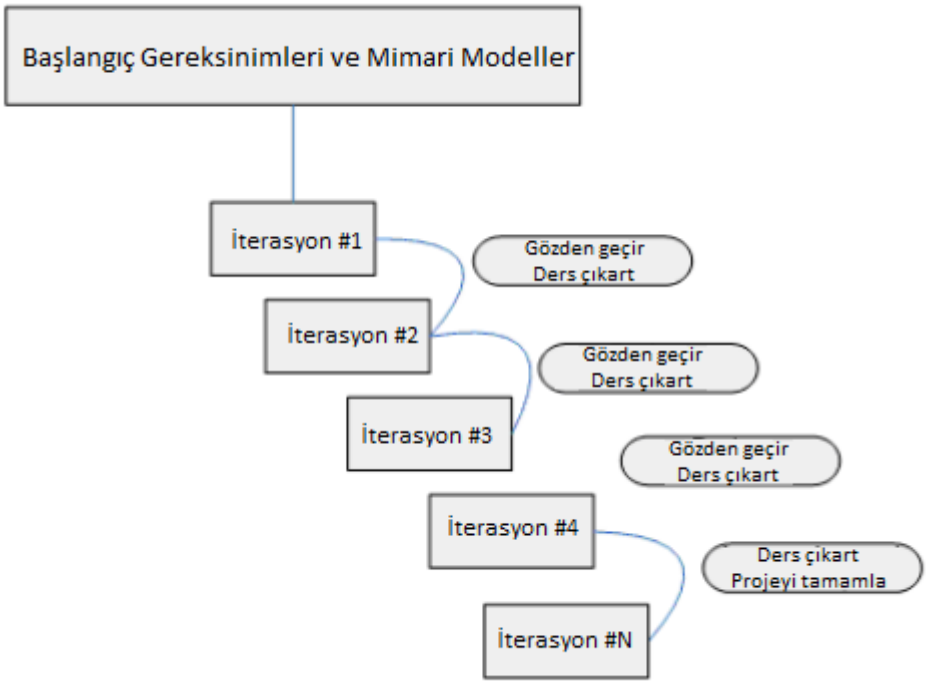
Uzun yıllar boyunca yazılım geliştirmek için en çok kullanılan metot olmasına karşın, Waterfall süreçleri sebebiyle başarısız olan birçok projenin ardından, yazılım geliştirme uzmanları geliştirmenin yapılışının ve planlamasının daha iyi bir yöntemini bulmak zorunda kalmışlardır. Bu arayıştaki en önemli odak noktası, "Yazılım geliştirme ekibinin daha fazla planlama yapması, daha fazla doküman yazması, değişime daha fazla karşı çıkması gerçekten daha iyi sonuç almaya yarayacak mı?" sorusudur. Birçok denemenin ardından bu metodu kullanarak daha iyi bir sonuç almayı başaramamış ve bir çok proje

geliştirmesi tamamlanamadan başarısız olmuştur. Tüm bu başarısızlıklar, "Scrum Metodolojisi"nin doğuşuna sebep olmuştur (P. Deemer, 2010).

Scrum metodolojisi, Agile (çevik) metodolojinin geliştirilmiş bir türüdür. Çevik metodoloji, yazılım geliştirme süreçlerini daha müşteri odaklı bir hale getirmiş ve ana hatları Agile manifesto ismiyle yayınlanmış olan dokümanda çizilmiştir (Mike Beedle, 2001).

Çevik yazılım geliştirmenin ana odak noktaları şu şekilde sıralanabilir:

- İsterlerdeki değişime ayak uydurabilmek
- Son kullanıcıyı mutlu etmek odaklı yazılım geliştirmek
- Yazılımı belli döngüler halinde müşteriye sunmak
- Süreçlere dahil olan herkesin yüksek motivasyona sahip olması
- Projenin gelişiminin, yazılımın performansı ile ölçülmesi
- Sürdürülebilir bir yazılım geliştirme sürecinin sağlanması
- Tasarıma ve teknik mükemmelliğe odaklanılması
- Tasarımın her zaman basitleştirilmesi
- Yazılım geliştirme ekiplerinin kendi kendine organize olması
- Projenin gelişiminin belli aralıklarla tartışılması



Şekil 2. Çevik yazılım geliştirme metodolojisi

Waterfall ile Çevik yazılım geliştirme metodunu karşılaştıracak olursak, her iki yöntemin de daha başarılı olabileceği senaryolar bulmak mümkündür. Waterfall; çokça dokümantasyonun olduğu, isterlerin net olduğu, görece daha küçük boyutlu projelerde daha başarılı olacaktır. Tüm isterlerin önceden belli olması durumunda yazılım geliştirme süreçleri çok daha basit olacaktır. Aynı bakış açısıyla, çevik yazılım geliştirme metodu, iyi bir takım iletişimi ve teknik anlamda yeterli yazılım geliştirme uzmanlarına ihtiyaç duyar. Buna karşın, süreçler esnektir ve büyük çaplı projelerde daha başarılı bir şekilde uygulanabilmektedir. Çevik yazılım geliştirme ile, yazılım geliştirme devam ederken isterlerde yaşanan değişikliklere tepki vermek ve uyum sağlamak daha kolaydır. Böylece boşa giden emeğin de önüne geçilmiş olur. İsterlerde oluşan değişiklikler hemen uygulanabilir durumdadır. Çevik yazılım geliştirme, müşteri ile yazılım geliştirme uzmanlarının yüz yüze görüşmesine ve detayları daha hızlı netleştirmesine imkân sağlar (A. Lawal, 2021). Günümüz rekabet koşulları, dijitalleşme, endüstri 4.0, yapay zekâ gibi bilgi teknolojilerindeki ilerlemeler, kullanım alanlarının yaygınlaşması, teknolojik yenilik ve dönüşümler iş süreçlerinin yeniden ele alınmasını, sürekli iyileştirmelerle optimize edilmesini zorunlu kılmaktadır (Şahinaslan E., 2023).

Scrum ise çevik yazılım geliştirme metodolojisinin günümüzdeki en başarılı uygulamalarından biridir. Scrum; yazılım geliştirmenin daha yenilemeli ve üst üste ekleyerek devam etmesine olanak sağlayan bir çerçeve çizer. Scrum'da, yazılım geliştirme "Sprint" ismi verilen döngüler halinde yapılır. Bu döngüler genellikle 2-4 hafta uzunluğundadır. Sprintler zaman sınırlıdır ve planlanan işin belirtilen tarihte bitip bitmemesinden bağımsız olarak kararlaştırılmış başlangıç ve bitiş tarihlerine sahiptir. Her sprintin başında takım bir araya gelerek bu sprint süresince hangi işlerin öncelikli olarak tamamlanacağına karar verir ve bu sürece "Planlama" ismi verilir.

Planlama sırasında direkt olarak müşteriden gelen isterler değerlendirmeye alınır ve yazılım geliştirme ekibi bu süreç içinde hangi işin yapılacağına karar verir. Daha başarılı bir planlama için, değerlendirmeye alınan her ister, teknik zorluğuna göre puanlanır ve yazılım geliştirme ekibi tarafından iş hacmine göre sprinte girip girmeyeceğine karar verilir. Bu tez çalışması süresince, planlama sürecinin parçası olan bu puanlama işlemi için, yazılım geliştirme ekibinin her isteri tek tek değerlendirip puanlamasından daha hızlı ve başarılı bir yöntem bulunmaya çalışılacaktır.

Bu amaç doğrultusunda, makine öğrenimi yöntemleri yardımıyla bir otomasyon sistemi tasarlanacaktır. Makine öğrenimi yöntemleri, yapılacak işin teknik karmaşıklığını göz önünde bulundurarak, kendisine verilen ister için doğru bir puanlama yapmayı amaçlayacaktır.

1.1 Problem

Günümüzde, Scrum birçok yazılım geliştirme firması tarafından yaygın olarak kullanılmaktadır. Bu yazılım geliştirme metodolojisinde, büyük özellikler kolayca kontrol edilebilmeleri için küçük parçalara bölünür. Yazılım geliştirme ekibi bir ister üzerinde çalışmaya başlamadan önce, isterin ekip tarafından puanlanması gerekmektedir. Bu amaçla, planlama toplantısı sırasında tüm ekip bir araya gelmeli ve her isteri tek tek incelemelidir. Bu incelemenin ardından, ekip isteri teknik karmaşıklığı açısından puanlayabilir. Bu planlama toplantıları her sprint için 8 saate kadar sürebilir (Jeff Sutherland, 2020).

Günümüz dünyasında, Scrum birçok yazılım geliştirme ekibi tarafından yaygın olarak kullanılmaktadır. Bu yazılım geliştirme metodolojisinde, büyük özellikler kolayca kontrol edilebilmesi ve uygulanmasının daha kolay olması amacıyla daha ufak parçalara bölünmektedir. Bu işlem aynı zamanda isterlerin yazılım geliştirme ekibi tarafından daha anlaşılır olması için de önemlidir. Yazılım geliştirme ekibi bir ister üzerinde çalışmaya başlamadan önce, isterin ekip tarafından incelenmesi ve puanlanması gerekmektedir. Bu amaçla, her sprint tekrar eden planlama toplantılarında tüm ekip bir araya gelerek her isteri tek tek incelemelidir. Ancak bu incelemenin ardından ekip isterleri puanlayabilir hale gelmektedir. Bu planlama toplantılarının süresi de her sprint için 8 saate kadar çıkabilmektedir. Uzunluğu sebebiyle, planlama toplantıları yazılım geliştirme süreçlerini yavaşlatmaktadır.

1.2 Amaç

Bu tez çalışmasında, Scrum metdolojisinin bir parçası olan planlama toplantılarında harcanan zamanın azaltılması hedeflenecektir. Planlama toplantıları sırasında, yazılım geliştirme ekibi, gelecek sprintte üzerine çalışılması hedeflenen her isteri tek tek inceleyip puanlayabilmek için çok fazla zaman harcamaktadır.

Makine öğrenmesi destekli otomasyon sistemi sayesinde, puanlama işlemleri için harcanan zaman azaltılacak ve yazılım geliştirme işinin yapılmasına daha fazla zaman kalacaktır. Bu iyileştirmeler sırasında, farklı dil modelleri kullanılarak, bu modellerin birbirine göre performansları ve başarı oranları da karşılaştırılacaktır.

1.3 Çalışmanın Önemi

Bu çalışma, Scrum yazılım geliştirme metodolojisinin bir parçası olan planlama toplantılarında, müşteriden direkt olarak gelen isterlerin tek tek incelenip puanlanması için harcanan zamanın azaltılarak, yazılım geliştirmeye daha fazla zaman ayrılabilmesini hedeflemektedir. Planlama toplantılarında harcanan zamanı azaltarak, yazılım geliştirme uzmanlarının yaratıcılığa ve ürüne daha fazla zaman ayırmaları sağlanacaktır. Buradan artan zaman sayesinde, aynı süre içinde daha fazla yazılım geliştirme işinin tamamlanması sağlanacaktır. Bunun bir sonucu olarak, süreçlerin yoğunluğundan kaynaklanan maliyetler azalırken, yazılım geliştirmeden elde edilen verimlilik artacaktır.

Ayrıca, Scrum'daki zorunlu faaliyetlerin neden olduğu dikkat dağınıklıkları ortadan kalkacağı için, yazılım geliştirme ekibi daha özverili ve dikkatli çalışmaya teşvik edilecektir.

1.4 Varsayımlar

Bu tez çalışması kapsamında, çok fazla zaman alan Scrum faaliyetlerinin bir makine öğrenmesi destekli otomasyon sistemi ile ortadan kaldırılmasıyla, Scrum yöntemlerinin yazılım geliştirme süreçlerine katkısı göz önünde bulundurularak, ekibin daha etkili ve verimli çalışabileceği varsayılmıştır. Ancak, Scrum faaliyetlerinin ekip üzerinde ne kadar olumsuz etkisi olduğu ve kazanılan zamanın ekibin verimliliği üzerindeki olumlu etkisinin ne kadar olduğu konusunda herhangi bir değerlendirme yapılmamıştır. Çalışma sırasında kullanılan veri setinin gerçek yaşam koşullarında her zaman aynı kalitede olmayabileceği ve yetersiz girdiler nedeniyle çalışmanın doğruluğunun azalabileceği göz ardı edilmiştir. Makine öğrenimi yöntemlerinin uygulandığı programlama dilleri arasında bir karşılaştırma yapılmamış ve her uygulamanın benzer verimlilikle çalışacağı varsayılmıştır. Benzer şekilde, hazırlanan algoritmaların çalışacağı işletim sistemlerinin performans etkisi de eşit olarak değerlendirilmiştir.

1.5 Limitler

Projenin amacı, planlama sırasında isterlerin puanlanması sürecini otomatikleştirerek ekibin harcadığı zamanı azaltmak olsa da, gerçek dünya uygulamalarında puanlanacak görevlerin yeterli olgunluğa ulaşmadan ekibe sunulması gibi bir kısıtlama vardır. Bu projede, planlama sırasında ekibe sunulan görevlerin yetersiz içeriğe sahip olma olasılığı ele alınamamaktadır. Bunun yerine, projede verimlilik ölçümleri için gerçek dünya senaryolarına uygun görevlerden oluşan bir veri seti seçilmiştir.

1.6 Tanımlar

Bu tez çalışmasında, makine öğrenmesi metotları ile çalışma konusundaki kolaylığı nedeniyle Python programlama dili tercih edilmiştir. Bu makine öğrenmesi algoritmaları ile, puan tahmini işinin her zaman olduğu gibi yazılım geliştirme ekibi yerine, makine öğrenmesi algoritmalarını kullanan bir otomasyon sistemi ile yapılması hedeflenmiştir.

Visual Studio Code yazılım geliştirme süreçlerinde tercih edilen geliştirme ortamı olmuştur.

Visual Studio Code: Visual Studio Code, yazılım geliştirmenin yapıldığı sırada kullanılan cihazın kaynaklarından verimli bir şekilde faydalanması ve kullanıcı dostu arayüzü sayesinde günümüzde en çok tercih edilen yazılım geliştirme ortamlarından biridir. Açık kaynaklı olarak geliştirilen bir platform olmasının dışında, kullanıcılar tarafından geliştirilen ve yazılım geliştirme süreçlerini çok kolaylaştıran birçok eklenti içermektedir. Bu eklentilerin bazıları kodun okunabilirliğini arttırırken, bazıları da veri setlerinin daha iyi görselleştirilmesine yardımcı olmaktadır.

Makine Öğrenmesi Metotları: Makine öğrenmesi metotları, yazılımları elimizde olan veriler ile eğitmemize imkân sağlayan algoritmalardır. Bu açıdan birçok imkân sunmaktadır. Makine öğrenmesi kendisine sunulan veri sayesinde kendini eğitebilir ve geliştirebilir. Makine öğrenmesi sayesinde; verileri işeyen, öğrenen, kendini geliştiren, seçim yapabilen ve sonuca varabilen yazılımlar oluşturmak mümkündür. Makine öğrenmesinin; denetimli öğrenme, denetimsiz öğrenme, yarı denetimli öğrenme ve pekiştirmeli öğrenme gibi farklı yöntemleri vardır.

Puan Tahmini: Puan tahmini Scrum'in bir parçası olarak yapılan, planlama sırasında takım tarafından değerlendirilen isterin teknik karmaşıklığının ölçülmesi işidir. İşin karmaşıklığı ve ortalama ne kadar efor gerektireceği ile ilgili takım olarak ortak bir karara varılması için yapılmaktadır. Bir ister puanlanırken, geliştirmenin yapılmasının zaman olarak ne kadar süreceğinden ziyade işin karmaşıklığına odaklanılır. Böylece şeffaflık ve görünürlük sağlanırken geliştirmenin sürekliliği güvence altına alınmış olur. Genel itibariyle, takımın sprint başında ne kadar iş alacağına ve hangi işi önceliklendireceğine karar vermesine yardımcı olur.

2. LİTERATÜR ARAŞTIRMASI

Makine öğrenmesi konusu derinlemesine incelendiğinde, günümüzde birçok problemin çözümünü makine öğrenmesi temelli yöntemlerle bulmanın mümkün olduğu görülmektedir. Bu çalışma kapsamında incelenen büyük dil modelleri ve yazılı metinlerin kategorizasyonu konusuna bakıldığında da yine hem akademik hem de ticari amaçlı birçok çalışma olduğu göze çarpmaktadır.

Yine tez çalışmasının bir parçası olan yazılım geliştirme süreçlerinin planlanması ile ilgili konular göz önünde bulundurulduğunda da aynı şekilde birçok akademik çalışmaya rastlamak mümkündür.

Bu iki ana başlık doğrultusunda ayrıntılı bir literatür araştırması yapılmış ve bu konuda bugüne kadar uygulanmış en başarılı yöntemlerin tespit edilmesine çalışılmıştır.

2.1 Makine Öğrenmesi ile Metin Kategorizasyonu

Uluslararası alanda bu konuda yapılan çalışmalar incelendiğinde ilk göze çarpan çalışmalardan biri 2002 yılında Fabrizio Sebastian tarafından yapılmış olan çalışmadır. Bu çalışmada mühendislik ile ilgili yazılmış metinleri konularına göre kategorilere ayırmayı hedeflemiştir ve bunun için de makine öğrenmesi tekniklerinden faydalanmıştır (Sebastiani, 2002).

Yine uluslararası alanda yapılan çalışmalardan bir başkası da Su Jin-Shu ile arkadaşlarının 2005'te yaptığı çalışmadır. Bu çalışmada, Jin-Shu ve arkadaşları, o güne kadar yapılmış olan çalışmalar sonucunda, metinlerin kategorize edilmesi konusunda bulunmuş yöntemleri derlemiş, süreç içinde bu alandaki gelişmelere ışık tutmuşlardır. Aynı zamanda gelecekte ilerlenmesi gereken ve gelişime açık olan konular hakkında da tespitlerde bulunmuşlardır (SU Jin-Shu, 2005).

Türkiye'de yapılan çalışmalara baktığımızda da yine bu konuda çok başarılı örneklere rastlamak mümkündür. Bunlara örnek olarak Emre Batuhan Baloğlu'nun yapmış olduğu Metin sınıflandırması için pekiştirmeli öğrenme: Politika-gradyan metotlarının farklı

topolojiler üzerinde değerlendirilmesi çalışması örnek verilebilir. Bu çalışmada Baloğlu, pekiştirmeli öğrenme yöntemleri ile IMDB'de yapılan yorumları olumlu ve olumsuz olarak sınıflandırmayı amaçlamıştır. Bu çalışmada makine öğrenmesi modellerinin hem verimlilikleri hem de zaman baslı olarak çalışma hızları karşılaştırılmıştır (Baloğlu, 2023).

2.2 Büyük Dil Modelleri Üzerine Yapılan Çalışmalar

Bir önceki bölümde en son bahsedilen çalışmalardan olan Emre Batuhan BALOĞLU, IMDB yorumlarının olumlu ve olumsuz olarak sınıflandırılması çalışmasında pekiştirmeli makine öğrenmesinin metotlarının yanında hazır önceden eğitilmiş dil modelleri de kullanmıştır. Örneğin bu çalışmasında DistilBERT ve RoBERTa dil modellerinin yapılan çalışma konusundaki verimliliklerini incelemiştir (Baloğlu, 2023).

Yine ülkemizde yapılmış bir başka çalışmayı incelediğimizde, Eyüp Halit Yılmaz'ın 2024 yılında yaptığı tez çalışması göze çarpmaktadır. Bu çalışmada açık kaynak kodlu olarak geliştirilen Mozilla projesinin yazılım hatalarının önceliklendirilmelerine bu tezde yapılana benzer bir şekilde otomatize bir sistem ile ulaşılmaya çalışılmıştır. Yılmaz yaptığı çalışmada 5 farklı kategorideki yazılım hatalarından eşit sayıda seçerek bir veri seti oluşturmuştur. Çalışma sırasında BERT dil modelinden faydalanarak 5 öncelik grubunda yazılım hatalarının sınıflandırılması üzerine uğraşmıştır (Yılmaz, 2024).

Yine benzer konuda, büyük dil modelleri ile biyomedikal konulu metinlerde yararlı bilgilerin çıkartılması üzerine 2020 yılında çalışma yapmış olan Nur Bengisu Çam'ın çalışmaları da literatürde önemli bir yer tutmaktadır. Bu çalışmada Çam, BERT tabanlı dil modelleri ile LLama dil modeli kullanılmıştır (Çam, 2020).

Metin sınıflandırma çalışmalarının globalde de birçok örneği bulunmaktadır. Bunlardan biri olan Aleksandra Edwards ile Jose Camacho-Collados'un "Language Models for Text Classification: Is In-Context Leraning Enough?" çalışmasını inceleyecek olursak, bu çalışmada araştırmacılar dil modellerini sınıflandırma yapılmasının beklendiği konularda eğiterek metin sınıflandırma işlemi üzerinde bu metodun başarısını test etmek istemişlerdir. Bu çalışmada yine LLama dil modelinden de faydalanan araştırmacılar, Twitter'da yazılmış olan yazılardan, haberlerde yer alan metinlerden, IMDB'de yapılmış

olan yorumlardan, medikal metinlerden ve kanuni metinlerden oluşan çok büyük bir veri seti ile çalışmışlardır (Edwards ve Camacho-Collados, 2024).

2.3 Scrum Süreçleri Üzerine Yapılan Çalışmalar

Scrum süreçleri üzerine yapılan çalışmalar incelendiğinde, yine hem yerli hem yabancı araştırmacıların yaptığı çok başarılı ve değerli çalışmalara rastlamak mümkündür.

Örneğin, Onur Erdoğan ve arkadaşları tarafından yapılan çalışmada, Scrum süreçlerinden olan Retro toplantısının verimliliğinin arttırılması konusunda istatistiksel araştırmalar yapılmıştır. Bu çalışmaya göre, Retro toplantıları sırasında; takımın iş hacmi, planlanmamış eforlar, efor hesaplaması sırasında bir puanın neye karşılık geldiği, takımın gerçek iş hacminin ne olduğu gibi konuların tartışılmasının, takımın gelişmesine katkıda bulunacağı ve Retro toplantılarının da daha verimli olmasının sağlanacağı gözlemlenmiştir (Onur E., 2017).

Direkt olarak bu çalışmada da hedeflendiği gibi Agile süreçlerin bir parçası olan Sprint Planlama toplantısının verimliliğini arttırmaya yönelik çalışmalar da elbette mevcuttur. Bunlardan en değerlilerinden biri de Matteo Golfarelli ve arkadaşları tarafından hazırlanan ve bir konferans çalışması olarak yayınlanan "Sprint Planning Optimization in Agile Data Warehouse Desing" çalışmasıdır. Bu çalışmada araştırmacılar, depolama üzerine geliştirilen bir projede Agile metotlarını uygularken, karmaşıklık hesaplamaları yapılırken, yapılacak bir işin değerini ölçerken, önceliklendirmeyi yaparken uygulanması ve göz önünde bulundurulması gereken önemli kriterleri belirlemeyi hedeflemişlerdir. Böylece planlama toplantılarının verimini arttırmakla kalmayıp bir Sprint süresinde yapılacak işlerden elde edilecek değeri de arttırmayı hedeflemişlerdir (Matteo Golfarelli, 2012).

2.4 Yapılacak İşlerin Puanlanması Üzerine Yapılan Çalışmalar

Bu çalışmada da odaklanılan yapılacak işlerin karmaşıklığı baz alınarak yapılan puan tahminlerinin geliştirilmesi konusu doğal olarak birçok başka araştırmacının da odak noktasında bulunmuştur. Joakim Zakrisson'un 2022'de yaptığı bir çalışmada, araştırmacı Agile süreçlerin en önemlisi olarak bahsettiği planlama toplantılarının verimini arttırabilmek adına, yazılım geliştirme uzmanlarının sürece katılımını arttıracak böylece

bu tahminlerin daha doğru yapılmasını sağlayacak yöntemler aramıştır. Yapılan çalışmada, toplam iş hacmi ve yapılacak işlerin karmaşıklığı hesaplamalarında tavsiye edilen yöntemlerden biri, bu tez çalışmasına da konu edilen, tüm yazılım geliştirme uzmanlarının aynı anda ve kapalı oylama ile puan tahmininde bulunduğu "Planlama Pokeri" yöntemidir (Zakrisson, 2022).

Yine planlama toplantılarında yapılacak işlerin puanlanmasının verimini arttırmak üzerine çalışma yapan bir başka ekip de Jayasri Angara ve arkadaşlarıdır. Bu çalışmada da araştırmacılar, planlama sırasında yapılan puan tahminlerinin tutarlılığını arttırmaya yönelik bir araştırmada bulunmuşlardır. Bu çalışmada da araştırmacılar Zakrisson'un yaptığı çalışmaya benzer şekilde en verimli yöntem olarak online erişim fırsatının da olduğu planlama pokeri yöntemini tavsiye etmişlerdir (Jayasri Angara, 2020).

3. GENEL BİLGİLER

Bu tez çalışmasının incelediği ana sorun, Scrum'ın zorunlu aktivitelerinden biri olan planlama aşamasında, isterler için puan tahmininin istenenden daha fazla zaman alıyor oluşudur.

Bu durumu daha anlaşılır kılabilmek için, Scrum Guide'da belirtilen aktiviteler ve tanımlamalara değinmek faydalı olacaktır. Scrum Guide'a göre, iki hafta süren bir sprint için ekip planlama ve düzenleme işlemlerine 8 saat harcar. Bu neredeyse bir tam iş gününe eşdeğerdir. Buna dayanarak planlama sürecinin ekibin iş gücünün neredeyse %10'unu oluşturduğu söylenebilir.

Bu tez çalışmasının amacı; yukarıda belirtilen yüksek çaba gerektiren ve zaman alıcı süreçleri kısaltmak, ekibin yazılım geliştirme ve değer üretmeye daha fazla zaman ayırmasına olanak sağlamaktır.

3.1 Agile

"Hızlı ve sürekli değişen iş ortamlarında, uyum sağlamak başarının anahtarıdır." Bu zamansız özdeyiş günümüz dünyasında giderek daha da önem kazanmaktadır. Çünkü, organizasyonlar yüksek rekabet, değişen pazar dinamikleri ve sürekli ortaya çıkan yıkıcı teknolojilerle karşı karşıyadır. Bu zorlukların ortasında başarılı olabilmek için, işletmeler giderek artan bir şekilde organizasyonel esneklik, duyarlılık ve yenilikçilik sağlamak amacıyla çevik metodolojilere yönelmektedir.

Çevik anlamına gelen Agile ismi, Latince yine çevik veya hızlı anlamına gelen "agilis" kelimesinden türetilmiştir. Agile, organizasyonların değişen gereksinimlere, müşteri ihtiyaçlarına ve piyasa koşullarına hızlı bir şekilde yanıt vermelerini sağlayan bir dizi prensip ve uygulamayı içermektedir. Yazılım geliştirmeden doğan Agile, başlangıçtaki sınırlarını aşarak günümüzde imalatta, pazarlamada, sağlık hizmetlerinde, finans alanında ve daha bir çok iş geliştirme uzmanlıklarında kullanılmaktadır.

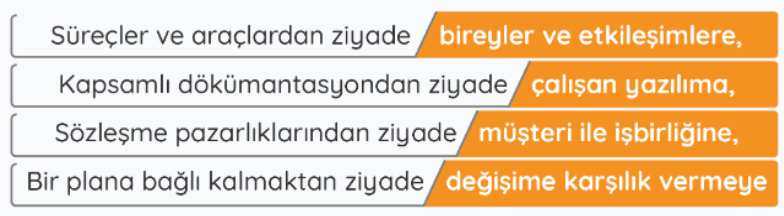
Agile, proje yönetimi ve ürün geliştirme için yenileyici ve esnek bir yaklaşımdır ve uyum sağlayabilmeyi, iş birliği ve sürekli iyileştirmeyi önemser. Hızla değişen iş ortamlarına ayak uydurma zorluğu çeken geleneksel, plan odaklı metodolojilere bir yanıt olarak ortaya çıkmıştır. Çeviklik, rekabet avantajı sağlamak için yenilik yapma ve yeni fırsatlar bulma konusunda gerekli varlıkları, bilgiyi ve ilişkileri hızlı ve şaşırtıcı bir şekilde toplama yeteneğidir (D'Aveni, 1994). Agile değişimi benimser, müşteri memnuniyetini önceliklendirir ve değeri aşamalı ve verimli bir şekilde sunabilmek için işlevler arası iş birliğini teşvik eder (S. Goldman, 1995).

Çeviklik, pazar arbitrajı fırsatlarını keşfetme ve onları kullanma süreciyle ortaya çıkan organizasyonel deneyimdir; yeni alternatiflerin araştırılması ve bilinmeyen fırsatlar odaklı rekabetçi eylemler için bilgi edinilir (Gürbüz, 2022).

Agile, bireyleri ve etkileşimleri önceliklendirir, değişimi benimser, değeri aşamalı olarak sunar, iş birliğini teşvik eder ve sürekli iyileştirmeyi destekler. Bu prensiplere bağlı kalarak, organizasyonlar belirsiz ve dinamik iş ortamlarında daha iyi yol alabilir, müşteri beklentilerini karşılayan yüksek kaliteli ürünler sunabilirler.

3.1.1 Agile manifestonun temel değerleri

Agile manifestonun 4 temel değeri bulunmaktadır. Şekil 3'te Agile'ın ana değerleri görülebilir.



Şekil 3. Agile'ın değerleri

- Süreçler ve araçlara göre bireyler ve etkileşim önceliklidir.
- Kapsamlı dokümantasyona göre çalışan yazılım önceliklidir.
- Müşteri ile iş birliği, sözleşme pazarlığından önceliklidir.
- Planlara bağlı kalmak yerine değişikliklere hızlı cevap vermek önceliklidir.

3.1.2 Agile manifestonun prensipleri

Agile manifesto, Figür 4'te görülen 12 başlıkta toplanabilecek prensipleri benimsemektedir (T. C. Krehbiel, 2017).



Şekil 4. Agile'ın prensipleri

3.1.2.1 Müşteriyi tatmin et

Değerli yazılım kavramı, müşterinin kullanabileceği ve önceliklendirebileceği kaliteli yazılım olarak kabul edilir. Yazılımın tamamlanan bölümlerini müşteriye erken ve sürekli olarak aktarmak ve müşteri memnuniyetini arttırmak için müşterinin görüşlerine başvurulmalıdır. Müşterilerden geri bildirim almanın en iyi yolu, yazılımın ilk çalışan sürümünü müşteriye mümkün olan en kısa sürede teslim etmektir. Böylece müşteri, geliştirilen yazılımın gerçekten hayal edildiği ve önceden konuşulduğu gibi çalışıp çalışmadığını bizzat test edebilir ve planlanan ilerlemede herhangi bir değişikliğin gerekip gerekmediğini gözlemleyebilir.

Yazılımın erken teslimatının dezavantajı, teslim edilen yazılımın hatalar içerebilmesidir. Teslimat öncesinde, yazılım ekibi müşteriye bazı hatalı davranışların olabileceğini detaylı bir şekilde açıklamalıdır, böylece beklenen özellikler mevcut olmadığında panik oluşmasını önleyebilir. Müşteriyle iş birliği yapan ve katı bir spesifikasyona bağlı olmayan bir ekip süreç boyunca gerekli değişiklikleri yapabilir. Bu, sürekli teslimat anlamına gelir (Alistair Cockburn, 2016) (Hunt, 2006).

3.1.2.2 Değişen gereksinimlere açık olmak

Agile'da geliştirme ekibi, ne kadar geç gelmiş olsa da isterlerdeki değişikliğe açık olmalıdır. Agile süreçleri müşterinin rekabet avantajı için isterlerdeki değişimden faydalanır (Mike Beedle, 2001). Buradan da anlaşılabileceği üzere, Agile uyum sağlayabilmeyi önemser. Geliştirme süreci devam ederken isterlerdeki değişikliklere ayak uydurabilmelidir. Bu değişiklikler müşteri için piyasada rekabet avantajı sağlayabilir.

Teslimat zamanı ve kaynakların, taleplere göre etkili ve doğru bir şekilde planlanması sağlanır. Müşteri açısından, geliştirme ekibi tarafından geliştirilen proje ilk seferinde mükemmel olmayabilir. Müşteri ile iletişim halinde kalarak, gözlemlenen hatalar hızlıca ve sıklıkla düzeltilmelidir. Yazılımın hangi aşamasında olursa olsun müşterinin değişiklik beklentilerinin gerçekleştirilmesi, müşteri memnuniyetini arttıracaktır (E. Cabello, 2017) (Hunt, 2006).

3.1.2.3 Müşteriye sık sık çalışan yazılım sunulması

İteratif geliştirme, erken ve sık geri bildirim alınmasına olanak tanır. Bu da yanlış geliştirme yapılması riskini azaltır (Iqbal, 2022). Agile, iteratif gelişimi önemser. Agile metodolojiler için, erken geri bildirim almak için küçük geliştirmeler ile çalışan yazılımların sürekli olarak müşteriye teslim edilmesi özellikle önemlidir.

Değişen gereksinimleri karşılamanın fikri projeler için heyecan verici ve faydalı olarak kabul edilir. Bununla birlikte, birçok ekip ve proje lideri bu fikirden memnun değildir. Onlar için bu değişiklikleri kabul etmek ve karşılamak karmaşık bir süreç olabilir ve kaotik bir atmosfer yaratırlar. Bu değişiklikleri kaosa yol açmadan yapmanın yolu, yazılımı paydaşlara sık ve sürekli olarak teslim etmektir. Ekip, düzenli olarak çalışan yazılımı müşterilere teslim etmek için periyodik olarak iterasyon yapar. Her iterasyonun sonunda, ekip her bir gelişmeyi müşteriye gösterir ve bu gösterimlerin retrospektif bir incelemesinden dersler çıkarır. Ardından, bir sonraki iterasyonda ne yapılması gerektiğine karar vermek için bir planlama toplantısı düzenlenir. Zaman sınırlı iterasyonlar, proje yöneticisinin ve geliştirme ekibinin geç gelen değişiklikleri ele almak için zamanı kontrol etmesine olanak tanır. Günlük incelemeler ve iteratif retrospektifler

ayrıca, bir sürümle ilgili problemlere yol açabilecek ciddi konuların erken tespitine olanak sağlar (Alistair Cockburn, 2016) (Karla Mendes Calo, 2010).

3.1.2.4 Günlük olarak iş birliği sağlanması

İş birimleri ve müşteriler, analistler, yazılım geliştiriciler, test uzmanları gibi ayrı rollerle bir ekip oluşturarak gereksinimleri karşılamalı ve sürekli iletişim halinde olmalıdır. Bu şekilde gereksinimleri doğru anlamak ve hızla karşılamak mümkün olacaktır. İş insanları ve ekipler proje boyunca günlük olarak birlikte çalışırlarsa, geri bildirimler daha hızlı alınır ve değişiklikler ve iyileştirmeler daha hızlı yapılabilir, bu da üretkenliği son derece artırır (Alistair Cockburn, 2016) (Karla Mendes Calo, 2010).

3.1.2.5 Motive edilmiş bireyler

Her ekip üyesi, bireysel yetenekler ve özelliklere sahip olarak kabul edilir. Kendi kendine motive olan ekip üyeleri ile iş birliği yapmanın, ekip performansı üzerinde olumlu bir etkisi vardır. Ayrıca, ekip üyelerinin kendi yetkinlikleri kapsamında atanan görevleri zamanında tamamlayacaklarına olan güven, motivasyonu artırmada önemli bir faktördür.

Çevik ekipler tarafından kullanılan teşviklerin bazı örnekleri şunlardır: yazılım geliştiricilere hiç hata olmadığında ödül verilmesi, testçilere rapor ettikleri yüksek sayıda hata için ödül verilmesi ve iş analistlerine ürettikleri belge ve açıklama sayısı için ödül verilmesi. Bu yaklaşım, yazılımda sık hatalar oluştuğunda yazılım geliştiricilere düşük performans puanı vermek yerine yüksek performans karşılığında ödüllendirmeye odaklanan bir teşvik sistemidir. Motive edici bir ekip ortamında herkes ihtiyaç duyduğu desteği alır ve proje başarılı olur (Alistair Cockburn, 2016) (Karla Mendes Calo, 2010).

3.1.2.6 Yüz yüze görüşmeler

Agile metodolojileri bireylere ve iletişimlere, süreçler ve araçlardan daha fazla önem verir (Mike Beedle, 2001). Bu cümleden de anlaşılabileceği üzere, Agile metodolojileri; efektif komünikasyonu ve tüm ekip üyeleri arasındaki iş birliğini, katı kurallarla sınırlandırılmış süreçler ve araçlardan daha önemli görür.

Agile ekipler için, belgelendirme sadece bir iletişim aracıdır. Belge üretmenin asıl amacı, düşünülen fikir hakkında doğru ve efektif bir şekilde iletişim kurulabilmesini sağlamaktır.

Belgelendirme önemli bir araç olmasına karşın iletişim kurmanın tek yöntemi de değildir. Yüz yüze görüşmeler her zaman fikirlerin paylaşılması ve beyin fırtınası yapılabilmesi için daha iyi bir yöntemdir. Agile, bireyler arasında güçlü bir iletişimi sağlamayı ve bu iletişimi belgelendirerek hatırlanabilir hale getirmeyi hedefler. İyi bir iletişimin sağlanması durumunda, ortak bir noktada buluşmak ve anlaşmazlıkların önüne geçmek çok daha kolay olmaktadır (Alistair Cockburn, 2016) (Rabia Saeed Malik, 2019).

3.1.2.7 İlerlemenin miktarının çalışan ürünler aracılığıyla ölçülmesi

Agile ekiplerin iteratif gelişimlerini çalışan yazılım ile sunmaları, gereksinimleri izlemek ve süreç ilerlemesini anlamak için müşterilere kısa aralıklarla üzerine çalışan projedeki gelişmeleri gözlemleme fırsatı tanır. Bu yaklaşım sayesinde, herkesin gelişim sürecinin farkında olması ve herhangi bir yanlış anlaşılmanın olmaması sağlanır (Alistair Cockburn, 2016) (Karla Mendes Calo, 2010).

3.1.2.8 Sürdürülebilir gelişimin teşvik edilmesi

Agile süreçler sürdürülebilir gelişimi teşvik eder. Sponsorlar, geliştiriciler ve kullanıcılar sürekli bir tempoyu kesintisiz olarak sağlayabilmelidirler (Mike Beedle, 2001). Yani, Agile'ın taraflarının amacı, sürekli bir yazılım geliştirme temposunu uzun vadede sağlayabilmek, takım üyelerinin tükenme noktasına gelmesinin önüne geçmek ve sürekli üretkenliği sağlayabilmektir.

Stabil bir yazılım geliştirme hızı yakalayabilmek ve yapılacak işlerin eşit olarak dağıtımını sağlamak, uzun süreli sürdürülebilirliğin anahtarıdır. Geleneksel yazılım geliştirme süreçlerinde, vadedilen tarih yaklaşıyor ve planlanan işler tamamlanamıyorsa, proje yöneticilerinin aklına gelen ilk çözüm geceleri ve hafta sonları ekstra çalışılmasıdır. Gerçekleştirilmesi mümkün olmayan bir planlama tarihi yüzünden, birkaç hafta yazılım geliştirme ekibi çok yüksek çıktı üretiyor gibi gözükse dahi, sonrasında takım içi üretkenlik ciddi oranda düşecektir. Uzun süre mesaili olarak çalışan ekipler, normal iş saatleri içinde çalışan ekiplere nazaran daha az üretebilmektedir. Bu yüzden, Agile takımları geliştirme hızlarını sabit ve sürdürülebilir tutmayı tercih ederler. Belirlenen sürede tamamlanabilecek maksimum geliştirmeyi yapmayı hedeflerler. Yenilemeli geliştirme ile, takım bir yıl içerisinde ne kadar iş tamamlanabileceğini tahmin etmeye

çalışmak yerine, birkaç hafta içinde ne kadar iş bitirebileceğini öngörme kabiliyetine sahip olmayı başarır. Bu sayede, yazılım geliştirme ekibi hem söz verdikleri işi zamanında tamamlar hem de ekstra saatlerde çalışma ihtiyacı duymaz (Alistair Cockburn, 2016).

3.1.2.9 Teknik mükemmelliğe sürekli dikkat

Yazılım geliştirme ekibi üyeleri, müşteriye sundukları yazılımların kalite beklentilerini sağlayabilmek ve müşteriyi memnun edebilmek için olabilecek en üst dikkati sergilemelidir. Agile takımlar, projenin başlangıcında çok büyük bir çıkış yapmak yerine, süreç içinde adım adım güvenilir ürünler ortaya koymayı hedefler. Ekip tasarımlarında ve kodlarındaki problemleri dikkatlice inceler ve bir sorun fark edildiğinde bu sorunları ortadan kaldırmak için gerekli olan zamanı ayırır. Böylece ilerleyen süreçte bu proje üzerinde hem geliştirmenin hem de bakımın yapılması kolaylaştırılmış olur (Hunt, 2006) (Karla Mendes Calo, 2010).

3.1.2.10 Basitlik esastır

Projenin bir parçası olarak, müşteri memnuniyetini artıran ve temel müşteri gereksinimlerini karşılayan eklemeler yapılmalıdır. Var olan bir yazılım projesinde mevcut koda yeni bir kod bloğu eklemek, mevcut kodu daha karmaşık hale getirir. Sistemler, nesneler ve hizmetler arasındaki bağımlılıklar kodu karmaşık hale getirir ve bir değişiklik istendiğinde bu değişiklik başka bir değişikliği gerektirir, bu da karmaşıklığın giderek artmasına neden olur. İsterleri olabildiğince basit şekilde uygulamak, bu kaostan kaçınmayı sağlar. Her iterasyonda yeni kod eklemek, kodu silmek ve yeni kod yazmaktan daha fazla zarar verir. Ek kodun bakım maliyeti, kodun silinip yeni kod yazılmasından her zaman daha yüksektir. Ekip, yazılımı küçük, bağımsız ve tek işlevli birimlerden oluşacak şekilde tasarlayarak kodu daha yönetilebilir ve bakımı daha kolay hale getirebilir. Bu yaklaşım aynı zamanda "tek sorumluluk" olarak da adlandırılabilir (Alistair Cockburn, 2016) (Karla Mendes Calo, 2010).

3.1.2.11 Kendi organize olan ekipler

Agile takımlarının kendi kendine organize olan ve birden fazla işlevi yerine getirebilen yapısı, kendi içlerinde daha iyi organize olmalarını, daha başarılı bilgi paylaşımı

yapmalarını ve yaptıkları işi sahiplenişlerini arttırır (Mike Beedle, 2001). Aynı zamanda, Agile takımları kendi içlerinde iş birlikçi olmalıdırlar. Agile takımlarında birbirinden farklı yeteneklere sahip insanların bir arada bulunmaları, ekip üyelerinin bireysel gelişimleri ve işi sahiplenişlerinin artışı için tavsiye edilmektedir.

Agile takımları kendi çalışma süreçlerini organize edebilen ve yapacakları işleri bağımsız planlayabilen kişilerden oluşmalıdır. Böylece, ekip yazılım geliştirmeyi yaptığı süreçte gelen yeniliklere açık olacak ve gerektiğinde süreçlerde değişiklik yapabilecektir. Bunun yanında Agile takımının üyelerinin yeni şeyler öğrenerek kendilerini geliştirmeye de açık olmaları gerekmektedir. Yazılım geliştirme başladığı andan itibaren, ekip isterler, tasarımlar ve mimari yapı hakkında ayrıntılı düşünmeli ve fikir alışverişinde bulunmalıdır. Bu işlem sırasında, projenin mimarları, projenin başında tüm gereksinimleri karşılayan mükemmel bir tasarım üretmeye çalışmak yerine, ekibin zamanla değişebilecek bir sistem tasarlamasına yardımcı olan teknikler kullanarak kademeli tasarım projeleri geliştirir (Hunt, 2006) (Karla Mendes Calo, 2010).

3.1.2.12 Düzenli olarak sürekli iyileşmeyi yansıtmak

İncele ve uyum sağla. Ekip performansını düzenli olarak değerlendirmeli ve verimliliği arttırmak için süreçleri düzenlemelidir (Mike Beedle, 2001). Ayrıca, sürekli iyileştirme, Agile metodolojiler için anahtardır. Agile yaklaşımlar, ekipleri düzenli olarak çalışmalarını değerlendirmeye, iyileştirme alanlarını belirlemeye ve verimlilik ve etkinliği artırmak için gerekli ayarlamaları yapmaya teşvik eder.

Bir ekip, yazılım geliştirme yöntemlerini sürekli olarak iyileştiriyorsa Agile olabilir. Agile ekipler, projenin nasıl işlediğini sürekli olarak gözden geçirir ve bu bilgiyi sonraki iyileştirmeler için kullanır. Ekip her gün toplandığında, iyileştirme ve değişiklik fırsatlarını arar ve fikir mantıklıysa, hemen değişiklik yapmaya başlarlar. Bir ekip olarak daha iyi hale gelmenin tek yolu, ekibin bugüne kadarki çalışmalarını değerlendirmek ve gelecekteki iyileştirmeleri planlamaktır (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).

3.1.3 Agile proje yönetiminin avantajları ve dezavantajları

3.1.3.1 Agile metotlarının avantajları

- Değişim her zaman memnuniyetle karşılanır. Geliştirme devam ederken projede değişiklik yapmak mümkündür ve kolaydır, bu nedenle Çevik'te planlama döngüleri kısadır. Haftalık değişiklikler, yapılacak işler listesi (backlog) yeniden düzenlenerek yapılabilir (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Nihai hedef belli olmasa dahi yazılım geliştirme yapmak mümkündür. Agile, varılmak istenen son nokta henüz belli değilken uygulanabilecek en iyi proje yönetim metodudur. Projenin geliştirilmesi devam ederken, projeden beklentileri ve isterleri değiştirmek mümkündür (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Geliştirme ekibinde güçlü etkileşim. Agile çalışma, sık iletişim ve yüz yüze görüşmelerin önemini vurgular. Birden fazla ekip birlikte çalışırken, Agile yaklaşım geliştiricilerin mevcut ilerlemeyi açıklamalarına ve projenin kendi bölümleri için sorumluluk almalarına olanak tanır (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Daha hızlı bir tempoda yüksek kaliteli çıktı. Projenin sürekli çalışan bir çıktı üretmesi, ekibin kaliteli geliştirme, test ve iş birliğine odaklanmasını sağlar. Her sprintte test yaparak, hatalar daha hızlı tespit edilip düzeltilebilir. Böylece yüksek kaliteli yazılım, tutarlı ve iteratif adımlarla daha hızlı teslim edilebilir (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Müşteriler de yazılım geliştirme sürecinin bir parçasıdır. Müşterilerin anlık olarak sonuçları gözlemleme, isterlerini paylaşma ve sonuçta ortaya çıkacak ürüne maksimum düzeyde etki etme şansı vardır. Geliştirme ekibi ile yakın çalışıyor olması, geliştirme sürecine karşı bir sahiplik hissi yaratır (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Geliştirilen ürün için iyileştirme süreklidir. Agile proje yönetimi, geliştiricilerin müşterilerden ve ekip üyelerinden sık sık geri bildirim almasına olanak tanır. Böylece hatalardan öğrenmek ve gelecekteki iterasyonları iyileştirmek mümkün olur (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).

3.1.3.2 Agile metotlarının dezavantajları

- Planlamalar daha kararsız bir hale gelir. Bazı durumlarda değişken isterler sebebiyle net bir teslimat tarihi belirlemek zor olabilir. Çünkü Agile metodolojiler, yinelemeli ve zaman sınırlı teslimatlara dayandığı ve proje yöneticileri sık sık görevleri tekrarladığı için, başlangıçta teslim edilmesi planlanan bazı öğeler zamanında hazır olmayabilir. Ayrıca, projede herhangi bir noktada yeni görevler eklenebilir ve bu da nihai hedef süresini uzatabilir (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Agile yöntemlerin başarılı bir şekilde uygulanabilmesi için ekibin her üyesinin kendi alanında uzman olması ve geniş çaplı bilgiye sahip olması gerekir. Agile ekipler genellikle 8-10 kişi gibi ufak gruplardan oluşur. Aynı zamanda seçilen Agile yöntem ve bu yöntemin gereklilikleri konusunda da geniş bir anlayışa sahip olunması gerekir (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Yazılım geliştirme uzmanlarının zaman temelli olarak yapılacak işleri değerlendirebiliyor olması gerekir. Agile yöntemler ancak tüm ekip tamamıyla ve sadece projeye adanmış olursa başarılı olabilir. Bunun için geleneksel proje geliştirme yöntemlerine nazaran çok daha zaman alıcı olan Agile süreçlerinin tamamına takımın dedike bir şekilde katılımının sağlanması çok kritiktir (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Belgelendirmeye fazla önem gösterilmez. Agile manifesto çalışan bir yazılımı ortaya koymayı dokümantasyondan daha çok önemser. Bunun sonucu olarak bazı takım üyeleri için yapılan işi dokümante etmek çok daha önemsizdir. Agile süreçler açısından dokümantasyon daha az önemli bir parça olsa dahi, dokümantasyon işlemleri ile yapılacak iş üzerine gerçekleşen teknik tartışmalar arasında bir denge kurmayı ve alınan kararlarla yapılan geliştirmeleri dokümante etmek de önemlidir (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).
- Başlangıçta alınan kararlar ve üzerine konuşulan fikirler, projenin geliştirilmesi sürecinde fazlasıyla değişebilir. Agile projeler başlangıçta net bir yol haritasına sahip olmayabilirler. Dolayısıyla geliştirme işlerinin sonunda elde edilen ürün, başlangıçta düşünülenden farklılıklar içerebilir. Çünkü, Agile projede çalışan ekibin geliştirme süreçleri üzerinde esnek hareket edebilmesini hedefler. Bu yüzden, ekip müşteriden gelen geri dönüşlere dayanarak farklı denemeler yapabilir ve süreçleri değiştirebilir.

Bu da başta hedeflenenden farklı bir sonuca varılmasına sebep olabilir (Samar Alsaqqa, 2020) (Sheetal Sharma, 2012).

3.1.4 Agile metodolojilerin sık uygulanan türleri

Agile süreçler, bu çalışmanın önceki bölümlerinde belirtilen öncelikler ve ilkeleri temel alarak manifestoyu kabul eden ve çalışma yöntemlerini çevik bir yaklaşıma göre geliştiren süreçler olarak tanımlanır. Zaman geçtikçe, endüstride birçok farklı türde çevik süreç denenmiş ve uygulanmıştır. Bu süreçlerin çoğu, çevik deklarasyondan sonra ortaya çıkmıştır. En bilinen ve en çok kullanılan çevik metodolojiler şu şekilde sıralanabilir: (Gürbüz, 2022)

- Kanban
- Dinamik Sistem Geliştirme Metodu
- Lean
- Ekstrem Programlama
- Scrum

Bu tez çalışması kapsamında, daha çok Scrum ve Scrum'ın etkinlikleri üzerine yoğunlaşan bir inceleme yapılacak. Aynı zamanda Scrum'a ait olan bu süreçlerde zorlayıcı ve zaman alıcı olan işlemler için, makine öğrenmesi temelli çözümler aranacak.

3.2 Scrum

Scrum, yinelemeli ve artımlı bir yaklaşımı takip eden bir Agile yazılım geliştirme metodolojisidir. Geliştirme sürecinde değişen müşteri gereksinimlerine hızlı uyum sağlamaya olanak tanır. Bu yaklaşım genellikle "ampirik yaklaşım" olarak adlandırılır çünkü sorunun önceden tamamen anlaşılabileceği veya tanımlanabileceği kabul edilmez. Bunun yerine, ekibin hızlı teslimat yapma ve ortaya çıkan gereksinimlere etkili bir şekilde yanıt verme yeteneğini en üst düzeye çıkarmaya odaklanılır.

Scrum, gereksiz adımları en aza indirerek ve daha önemli işleri tamamlayabilmek için üretken zamanı en üst düzeye çıkararak kolay uygulanabilir bir süreç olacak şekilde tasarlanmıştır. Anlaşılması kolaydır, ancak ustalaşmak için önemli bir uzmanlık gerektirir (Leferve-Ardant, 2008).

3.2.1 Scrum tarihçesi

Scrum süreci, ilk olarak Jeff Sutherland ve Ken Schwaber tarafından 1995 yılında ABD'nin Austin kentindeki Oopsla konferansında tanıtıldı. "SCRUM Yazılım Geliştirme Süreci" adlı bir makale yayımladılar ve "Scrum" terimini Takeuchi ve Nonaka'nın "Yeni Ürün Geliştirme Oyunu" adlı makalesinden türettiler. Araştırmaları, küçük ve kendi kendini organize eden ekiplerin görevler yerine hedefler verildiğinde karmaşık ürünlerin etkili bir şekilde geliştirildiğini ortaya koydu. En iyi ekipler, rehberlik sağlanırken ortak hedeflerine ulaşmak için kendi stratejilerini geliştirme esnekliğine sahip olan ekiplerdir.

2001 yılında Jeff ve Ken, Agile Yazılım Geliştirme Manifestosu'nu oluşturan liderler arasında yer aldı. 2002 yılında Ken Schwaber, Mike Cohn ve Esther Derby ile, son derece başarılı Sertifikalı Scrum Master'lar sağlamak amacıyla Scrum Birliği'ni kurdu. Jeff Sutherland, 2006 yılında kendi şirketi olan Scrum Inc.'i kurdu ve sertifikalı Scrum kursları sunmaya devam etti. Ken, daha sonra 2009 yılında Scrum'un kalitesini ve etkinliğini daha da artırmak için Scrum.org'u kurdu.

2010 yılında Scrum Kılavuzu'nun yayımlanması ve ardından 2011 ve 2013 yıllarında yapılan güncellemeler, Jeff ve Ken'in çalışmalarını Scrum için küresel olarak tanınan bilgi kaynağı olarak sağlamlaştırdı. Yıllar içinde Scrum, dünya genelindeki yazılım geliştirme şirketleri tarafından geniş çapta benimsendi ve Agile yazılım geliştirme için en çok kullanılan çerçeve haline geldi.

Günümüzde Scrum, yalnızca yazılım geliştirme ile sınırlı kalmayıp, üretim, pazarlama, operasyonlar ve eğitim gibi çeşitli alanlarda da uygulama bulmuştur (Şahin, 2016).

3.2.2 Scrum teorisi

Scrum, deneyimler üzerinden bilgi edinmeyi ve elde olan bilgiler ile seçimler yapmanın önemini benimseyen, deneysel bir süreçtir (Jeff Sutherland, 2020). Her Scrum uygulamasının vazgeçilmez üç faktörü vardır: Şeffaflık, Denetleme ve Adaptasyon.

Şeffaflık: Hedeflenen sonuca ulaşabilmek için, yazılım geliştirme sürecinin önemli parçaları tüm paydaşlar tarafından gözlenebilir olmalıdır. Yazılım geliştirmenin süreci ve

bulunulan son durumu herkese anlatabilmek için ortak bir dilde herkes ile buluşabilmek önemlidir.

Denetleme: İstenmeyen değişikliklerin önüne geçebilmek ve Scrum hedefinin gerçekleşeceğinden emin olabilmek adına tüm paydaşların Scrum ürünlerini ve süreçlerini denetleme şansına sahip olmalıdır.

Adaptasyon: Gözlemcilerden herhangi birinin Scrum süreçlerinin ve ürünlerinin bir veya birden fazlasında bir sapma gözlemlemesi durumunda, Scrum ekibinin gerekli değişiklikleri sağlayarak ve yeni sürece uyum sağlayarak ileride yaşanabilecek daha büyük sapmaların önüne geçmesi gerekmektedir.

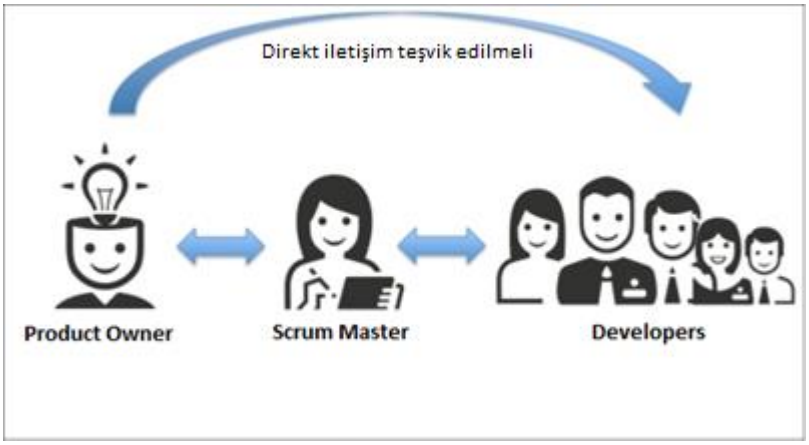
Scrum, etkinlikler bölümünde de anlatıldığı üzere, denetleme ve adaptasyonu mümkün kılmak için dört ana etkinlik içermektedir. Bu etkinlikler: Sprint Planning, Daily Scrum, Sprint Review ve Sprint Retrospective olarak sıralanabilir.

3.2.3 Scrum rolleri

Bir Scrum ekibinde üç ana rol bulunmaktadır: Geliştirmeciler (Developer), Scrum Lideri (Scrum Master) ve Ürün Sahibi (Product Owner).

Scrum ekipleri kendi kendine organize olan ve çok yönlü ekiplerdir. Kendi kendine organize olabilen ekipler, yaptıkları işte başarıyı yakalamak için ne yapmaları gerektiğine de kendileri karar verebilirler. Çok yönlü ekipler de üzerinde çalıştıkları işi başarı ile ve dışarıdan yardım almadan tamamlayabilmek için gerekli olan tüm özelliklere sahiptirler.

Scrum ekipleri tekrar eden denemeler halinde ve deneysel şekilde ürün geliştirirler. Ortaya çıkan ürünler, geliştirilen ürünün kullanılabilir bir hali olmalıdır.



Şekil 5. Scrum Ekibinde Roller

Ürün Sahibi (Product Owner (PO)): PO Scrum ekibi tarafından üretilen değerden sorumludur ve PO'nun temel hedefi bu üretilen değeri maksimize etmektir. PO ancak bir kişi olabilir, bir komite tarafından bu görev yerine getirilemez. PO, takıma karşı müşterileri ve diğer paydaşları temsil eder.

PO aynı zamanda ürüne dair yapılacak işler listesini (Product Backlog) kontrol etmekle yükümlüdür. Bu iş listesi, ürünün son halinde olması beklenen özellikler ve fonksiyonalitelerin kısa ama net bir şekilde anlaşılabilir açıklamalarını içerir. Aynı zamanda PO bu iş listesinde bulunan her bir iş maddesini takıma açıklamak ve anlaşılır kılmaktan da sorumlu olan kişidir. PO müşterilerden ve paydaşlardan gelen geri bildirimlere göre yapılacak olan işlerin önceliklendirilmesini değiştirme özgürlüğüne sahiptir.

Yazılım geliştirme uzmanları üstlendikleri işleri tamamladıklarında ve ürünün planlanan geliştirme süreci bittiğinde, PO ürünün son halini paydaşlara duyurur. PO yapılacak işler listesinin gözden geçirilip düzenlenmesi işini yapabilir ya da takımdan birinin yapmasını sağlayabilir. Takımdan bir başkası bu düzenlemeyi yapıyor olsa dahi, iş listesinden her zaman PO sorumludur.

Bir organizasyonda başarının sağlanabilmesi için, tüm organizasyonun, bireylerin aldığı kararlara saygı duyması önemlidir. Geliştirme ekibine ne yapacağını kimse dikte edemez

ve aynı şekilde yazılım geliştirme ekibi de iş birliğinden uzak bir şekilde çalışma metodu geliştiremez (Şahin, 2016).

Yazılım Geliştirme Ekibi: Yazılım geliştirme ekibi, her sprint sonuna bitmiş bir ürün çıkartmak ile sorumlu olan profesyoneller grubudur. Bağımsız olarak geliştirme yapmalarına ve başka bir takıma ihtiyaç duymadan ürün geliştirmelerine imkân tanıyacak şekilde farklı yeteneklere sahip ve kendi kendine organize olabilen bir ekiptir. Ekip üyelerine, yazılım geliştirme uzmanı (developer) denir. Ekibin üyelerinin, analist ya da testçi gibi spesifik rolleri yoktur. Hepsi yazılım geliştirme uzmanı olarak geçer. Her bireyin uzmanlaştığı farklı bir alan olabileceği gibi, ürünün geliştirme süreçlerinden ve ortaya çıkan üründen takımın tamamı bir bütün olarak sorumludur.

Takım büyüklüğü genellikle 3 ile 9 kişi arasında seçilir. Bu seçim sırasında hem çevikliğe hem de gerekli işlerin tamamlanabilmesi için ihtiyaç duyulan iş gücünün sağlanmasına önem gösterilir. Daha ufak takımlar beklentileri karşılayacak hızda ürün geliştirmesinde zorlanırken, daha büyük takımlar ise koordinasyon konusunda problem yaşayabilirler. PO ve SM, aktif olarak iş listesinden iş alıp, geliştirmeye katkıda bulunmadıkları sürece takım boyutu hesaplanırken dahil edilmezler (Şahin, 2016).

Yazılım geliştirme ekibi sprint bazlı yapılacak iş listesinden sorumludurlar. Ancak bu ekip sprint sırasında yapılacak iş listesinde değişiklik yapabilir. Aynı zamanda sprint planlaması sırasında gelecek sprintte hangi işlerin yapılacağına, PO'nun önem sırasına da dikkat ederek, yazılım geliştirme ekibi karar verir. Yapılacak işler konusunda "ne" yapılacağına PO karar verirken, "nasıl" yapılacağına ise ancak yazılım geliştirme ekibi karar verebilir. Tüm teknik sorumluluk tamamen yazılım geliştirme ekibine aittir.

Scrum Lideri (Scrum Master (SM)): Scrum Lideri, Scrum süreçlerinin ve sorumluluklarının tüm Scrum ekibi tarafından ayrıntılı olarak anlaşılması ve uygulanmasından sorumludur. Scrum prensiplerini, bilinen en iyi uygulamaları ve Scrum kurallarını önceliklerler.

Dikkat edilmelidir ki; Scrum lideri bilinen tarzda klasik bir proje yöneticisi ya da takım lideri değildir. Tam aksine, takım içinde işleri kolaylaştırıcı, konsantrasyonu sağlayıcı ve

üretilen ürünün sahip olduğu değeri yükseltmeye kendini adamış bir lider konumundadır (Şahin, 2016).

SM'in PO'ya, yazılım geliştirme ekibine ve parçası olduğu organizasyona karşı farklı sorumlulukları vardır.

SM'in PO'ya karşı sorumlulukları:

- Yapılacak iş listesinin yönetilmesinde uygulanabilir başarılı yöntemler bulmak.
- Yapılacak iş tanımı kapsamındaki isterlerin Scrum ekibi tarafından anlaşılabilmesine yardımcı olmak.
- Çeviklik kavramını anlamak ve uygulamak.
- İhtiyaç duyulan ve istenen scrum etkinliklerini düzenlemek.
- Ürün planlama sürecinin deneysel bir yaklaşım içerdiğinden emin olmak.

SM'in yazılım geliştirme ekibine karşı sorumlulukları:

- Takımın üretkenliğini, fonksiyonalitesini ve motivasyonunu yükseltmeye çalışmak.
- Yazılım geliştirme ekibinin kendi kendine organize olabilmesi ve farklı yetenekleri içeren bir ekip haline gelebilmesine koçluk etmek.
- Ürünün geliştirme sürecinde yazılım geliştirme ekibi açısından süreklilik arz eden, ürün geliştirme süreçlerinin ilerlemesine ket vuran, sorunları, problemleri ve özellikle bağımlılıkları ortadan kaldırmak.

SM'in parçası olduğu organizasyona karşı sorumlulukları:

- Organizasyonun Scrum'a geçiş sürecine liderlik etmek
- Çalışanların ve paydaşların Scrum'ı anlamasına ve deneysel bir şekilde uyum sağlamalarına yardımcı olmak.
- Organizasyon içindeki diğer SM'ler ile ortaklaşa çalışarak, organizasyondaki Scrum'ın uygulamalarını daha efektif hale getirmek.

İyi bir SM'in sahip olması gereken altı özellik vardır. SM; sorumluluk sahibi, mütevazi, işbirlikçi, adanmış, etkili ve bilgili olmalıdır (Cho, 2009).

3.2.4 Scrum etkinlikleri

Tüm Scrum etkinlikleri maksimum süreye sahip, yani zaman sınırlıdır. Bu yaklaşım sayesinde, tutarlılık artar ve sonu belirsiz, uzun toplantılara ihtiyaç da ortadan kalkar. Bir örnek olarak, Daily Scrum 15 dakika ile sınırlı bir toplantıdır.

Scrum açısından kısaltılması mümkün olmayan tek etkinlik Sprint'tir. Bir Sprint başladığı anda, süreyi kısaltmanın bir yolu yoktur. Bunun haricinde, Scrum kapsamındaki tüm diğer etkinlikler, hedefine ulaştığı durumda kısaltılabilir.

Scrumdaki her etkinli, üretkenliği arttırabilmek için adaptasyonun ve denetlemenin sağlanması açısından önemli bir şanstır.

3.2.4.1 Sprint

Sprint, Sprint Planning, Daily Scrum, geliştirme işlemlerinin yapılması, Sprint Review ve Sprint Retrospective gibi tüm diğer etkinlikleri kapsayan ana bir etkinliktir. Süre sınırı bir ay ya da daha sık uygulanan yöntem olarak iki haftadır. Uzun süreli iletişimsizlik sebebiyle ortaya çıkabilecek kompleksiteleri ve riskleri ortadan kaldırabilmek adına bir Sprint'in uzunluğu hiçbir zaman bir ayı geçmemelidir.

Sprint'in süresi başlamadan önce belirlenmelidir ve başladıktan sonra da değiştirilmemelidir. Her Sprint'in süreç boyunca korunması gereken bir "Sprint Hedefi" bulunur. Her Sprint'in sonunda da kullanılabilir ve müşteriye göndermeye hazır bir ürün ortaya çıkarılmalıdır.

Müşteri ve diğer paydaşlar ile yapılan görüşmeler sırasında eğer Sprint hedefi anlamsız hale gelir ya da başarılmasının mümkün olmadığı anlaşılırsa, ancak PO'nun Sprint'i iptal etme yetkisi vardır. Sprint iptalleri çok sık görülmeyen ve Scrum ekibi açısından rahatsız edici olaylardır. Sprint iptali halinde Scrum ekibinin tekrar bir araya gelerek yeni bir hedef ile yeni bir sprintin planlamasını yapmaları gerekir (Şahin, 2016).

3.2.4.2 Sprint planlaması

Her Sprint'in başlangıcında, Scrum ekibi, gelecek sprinte yapılacak işleri planlamak üzere bir Sprint planlamasında (Sprint Planning) bulunur. Bu planlama sırasında sprint bazlı yapılacak işler listesi ve Sprint hedefi belirlenir.

Sprint planlaması genellikle bir ay süren sprintler için sekiz saat olmak üzere zaman sınırlı etkinliklerdir. SM, bu etkinliği düzenlemek, tekrarlanmasını sağlamak ve katılımcıların bu etkinliğin amacını anlamasın konusunda yol gösterici olurken, zaman sınırlamasına da riayet edilmesini sağlamakta çok önemli bir rol oynar.

Sprint planlaması sırasında, yazılım geliştirme ekibi, belirlenen sprint hedefini başarı ile tamamlayabilmek için yapılması gereken işlerden, sprint bazlı yapılacak işler listesi oluşturur. Sprint süresince yapılacak işlerin anlaşılır ve net bir listesi bu sayede oluşturulmuş olur.

Sprint planlaması sırasında takım, "Gelecek Sprint sırasında hangi iler tamamlanabilir?" ve "Beklenen isterleri sağlayan bir ürün ortaya koyabilmek için ne kadar iş yapılması gerekmektedir?" gibi sorulara cevap arar.

Hangi geliştirmelerin yapılabileceği ile ilgili soruyu cevaplayabilmek için, yazılım geliştirme ekibi kapasitesini de göz önünde bulundurarak işleri seçer. Böylece Sprint bazlı yapılacak işler listesi oluşturulmuş olur. Bu iş seçim işlemi sırasında, yazılım geliştirme ekibi aldığı işleri Sprint sonunda tamamlanmış halde teslim edeceğine söz vermiş olur (Şahin, 2016).

3.2.4.3 Günlük toplantı

Daily Scrum 15 dakika ile sınırlı olmak üzere, takımın günü planladığı toplantıdır. Bu toplantı sırasında yazılım geliştirme ekibi bir önceki gün neler yapıldığını ve bulunulan günde nelerin yapılması gerektiğini tartışır. Toplantı her gün aynı zamanda aynı yerde yapılır. Bu toplantı yalnızca yazılım geliştirme ekibine aittir. SM ve PO toplantıya ancak dinleyici olarak katılabilir ancak toplantıya müdahale edemez (Gürbüz, 2022).

Daily Scrum sırasında, yapılan iş ile ilgili bir problem, engel, risk ortaya çıkarsa SM konuyu not alır ve gerektiğinde bunu Scrum Board'a yansıtır.

Scrum Master yazılım geliştirme ekibinin toplantıyı 15 dakika ile sınırlı tuttuğundan emin olmakla görevlidir.

3.2.4.4 Sprint review

Bir aylık bir Sprint için, Sprint Review dört saat ile sınırlı bir toplantıdır. Daha kısa sprintler için genellikle bu toplantılar daha kısa sürebilir (Kır, 2014).

Sprint Review sırasında, Scrum ekibi ve diğer paydaşların Sprint süresince tamamlanmış olan işlerin son halini gözlemleme fırsatları olur. Product Owner son ürünü sunar ve bu sırada neyin tamamlanmış ve neyin de tamamlanamamış olduğunu açıklayarak paydaşlardan geri bildirim toplamayı hedefler. Yazılım geliştirme ekibi de sunulan ürün ve anlatılan fonksiyonaliteler ile ilgili yöneltilen sorulara cevap verir.

Sprint Review özellikle tüm paydaşlara yazılım geliştirme ekibinin Sprint boyunca yaptıklarını sunup, üzerine konuşulan ürün hakkında canlı ve anlık geri bildirim toplayabilmek adına çok önemlidir. Bu, Scrum ekibine, üzerinde çalıştıkları ürünü kontrol etme ve gelen geri bildirimlere göre adaptasyon geçirerek gelecek Sprintlerde yapacakları işleri planlama şansı tanır (Şahin, 2016).

Sprint Review sonrasında, PO yapılacak işler listesini gözden geçirir ve gerekli olan değişiklikleri yapar. Böylece tüm katılımcılar ürünün geliştirilmesinde bir sonraki adım hakkında söz sahibi olmuş olur. Bu iş birliği, bir sonraki Sprint planlamasında dikkate alınacak öğelerin ortaya çıkmasını sağlar.

3.2.4.5 Sprint retrospective

Sprint Retrospective, Sprint'in bitiminde yer alan etkinliktir. Bir ay süren Sprintler için üç saat ile sınırlı iken, daha kısa süreli Sprintlerde daha kısa sürebilir. Scrum Master toplantının düzenleyicisi ve moderatörü olarak toplantıda bulunurken, aynı zamanda aktif olarak toplantıya katkıda bulunup bir takım üyesi olarak Scrum süreçlerinin doğru uygulandığından emin olunmasını sağlar (Şahin, 2016).

Sprint Retrospective, yazılım geliştirme ekibine, uyguladıkları süreçleri ve gelişimlerini gözden geçirme, böylece gerekiyorsa ihtiyaç duyulan adımları atarak gerekli

değişiklikleri yapma fırsatı sunar. Scrum Master toplantıyı düzenleyerek katılımcıların bu toplantının önemini ve amaçlarını anladıklarından emin olur.

Bu toplantının önemi:

- Geçmiş Sprinti gözden geçirmek.
- Sürekli süreç gelişimleri üzerine konuşup atılacak adımları belirlemek.

Sprint Retrospective sırasında, Scrum Master mümkün olabilecek gelişime açık konular ile ilgili notlar alır ve bunlardan eylem öğeleri oluşturur. Her eylem öğesinin bir tanımı ve bir de sorumlusu olan kişi vardır. Bir sonraki Sprint Retrospective, bir önceki Retrospective toplantısında oluşturulan tüm eylem öğelerinin son durumlarının konuşulması ile başlar (Gürbüz, 2022).

3.2.5 Scrum araçları

Scrum araçlarının amacı, var olan bilginin herkes tarafından kolayca ulaşılarak ihtiyaç duyulan şeffaflığın yaratılmasıdır. Bu araçlar aynı zamanda Scrum süreçlerinin denetlenmesini ve gerekli olan durumlarda uyarlanmasını sağlar. Scrum, dört adet araç tanımlar: Product Backlog, Sprint Backlog, Increment ve Definition of Done (Gürbüz, 2022).

3.2.5.1 Product backlog

Product Backlog, başarılı ve müşteriye göndermeye hazır bir ürün geliştirebilmek için ihtiyaç duyulan isterlerin, özelliklerin, hataların, ek özellik beklentilerinin önceliklendirilmiş ve sıralanmış bir listesidir. Product Owner, Product Backlog'un düzenlenmesinden ve bakımından sorumludur. Yazılım geliştirme ekibi için Product Backlog'un hazır ve erişilebilir olmasını sağlamak da PO'nun görevidir.

Product Backlog sürekli olarak değişen ve gelişen bir araçtır ve ürünün geliştirilmesi süreci devam ettiği sürece, hiçbir zaman tamamlanmış olarak kabul edilmez. Ürün var oldukça, ürünün ihtiyaçları ve isterleri sürekli olarak Product Backlog'a yansıtılır. Product Backlog'taki her madde Product Backlog Item (PBI) olarak tanımlanır. Her PBI'ın net bir tanımı, değeri, boyut tahmini ve kabul kriterleri bulunmalıdır.

PBI'ların boyut tahminlerini yazılım geliştirme ekibi yaparken, Product Owner ekibin üzerinde çalıştığı ve boyut tahmini yapacağı konuyu anlamalarına destek olabilir. Boyut tahmininde yine de son söz sahibi yazılım geliştirme ekibidir. Bu boyut tahminleri genellikle Fibonacci serisi kullanan "Story Point"ler ile belirlenir. Bu tahminlerin tutarlılığı için ekip önceden bir referans PBI belirler ve yeni boyut tahminlerini bu refernası göz önünde bulundurarak yaparlar (Kır, 2014).

3.2.5.2 Sprint backlog

Sprint Backlog, yazılım geliştirme ekibi tarafından belirlenmiş olan, Sprint süresince üzerine çalışılması hedeflenen işler listesidir. Sprint sonunda ortaya çıkacak üründe bulunması gereken özellikler ve Sprint Hedefi'ni mümkün kılmak için yapılması gerekenleri içeren bir listedir.

Sprint Backlog'a eklenecek olan PBI'lar yazılım geliştirme ekibinin iş hacmine bağlı olarak seçilmelidir. Takımın bir Sprintte başa çıkabileceğinden daha fazla iş eklenmemelidir, çünkü bu takımın iş hacmini olumsuz olarak etkileyecektir. Sprintte üzerine çalışılacak maddeler seçildikten sonra yazılım geliştirme ekibi bu maddelerin tamamlayıp Sprint sonunda teslim edilebilmesi için ihtiyaç duyulan görevleri belirler, boyut tahmininde bulunur ve Scrum Board üzerinden takibini yapabilir. Sprint Backlog'taki işlerin hiçbiri direkt olarak takımdan birilerine atanmaz. Tam tersine takım üyeleri kendi kendine organize olabilme özelliği ile öncelikli olan işlerden başlayarak yapılacak işleri kendileri üzerlerine alırlar.

Sprint başladıktan sonra Sprint Backlog'ta değişiklik yapma hakkı yalnız yazılım geliştirme ekibine aittir. Sprint Backlog, Sprint süresince planlanan işlerin ne aşamada olduğu ve yazılım geliştirme ekibinin işlerinin nasıl gittiğinin gözlemlenebilmesi için büyük öneme sahiptir. Bu sayede gerçek zamanlı olarak şeffaflık sağlanmış olur.

Sprint süresince işlerin gelişim süreçlerinin takibi için Sprint Burn-Down tablosu kullanılır. Bu tablo herkese açık olarak sunulur ve Sprint sonuna kadar bitirilmesi hedeflenen işlerin ne kadarının tamamlandığını ve ne kadar yapılması gereken iş kaldığını gösterir (Şahin, 2016).

3.2.5.3 Bitmiş iş tanımı

Bir PBI'dan bahsedildiği sırada, işin bitmiş olması ile ilgili herkesin kafasında farklı bir tanımlama ve ister listesi oluşabilir. Bu tanımlamalar ve isterler listesine bitmiş iş tanımı (Definition of Done (DoD)) adı verilir. DoD her Scrum ekibi için değişiklik gösterebileceği gibi, takım içindeki üyeler arasında bu konuda net ve ortak bir anlayış olması çok önemlidir (Kır, 2014).

Genel olarak DoD aşaüıdaki gibi kriterler içermelidir:

- İsterlerin analizinin bitmiş olması
- Tüm dokümantasyon işlerinin bitmiş olması
- Diğer takım üyeleri tarafından kontrolünün yapılmış olması
- Yazılım geliştirmenin bitmesi
- Unit testlerin yazılması
- Kalite kontrol testinden başarıyla geçmesi
- Kullanıcı kabul testinden başarıyla geçmesi
- Hiçbir hata içermemesi
- Paydaşlara ve temsilcilere başarılı bir şekilde sunulmuş olması
- Ürüne başarılı bir şekilde entegre edilmesi

3.2.5.4 Increment

Increment, tanım olarak; Sprint boyunca yapılan tüm işlerin toplu bir birleşimi olarak nitelendirilebilir. Sprint sona erdiğinde, Scrum ekibinin DoD tanımına uygun olacak şekilde bir Increment üretilmiş olması gerekmektedir. Bu da, PO bu Increment'i müşterinin kullanımına sunmaya karar verse de vermese de, kullanılabilir durumda olması, isterlerin tamamını karşılaması anlamına gelir (Şahin, 2016).

3.3 Makine Öğrenmesi

Öğrenmenin İngilizcede "öğrenmek", "beceri kazanmak", "yetenek kazanmak" ve "bilgi edinmek" gibi çeşitli anlamları vardır. Öğrenme aynı zamanda deneyim, eğitim ve iş yoluyla davranışta meydana gelen ölçülebilir ve nispeten kalıcı değişikliktir. Konu öğrenmeye gelince, bir insan ya da evcil bir hayvan yerine bir makinenin öğrenebileceğini

düşünmek ilk bakışta zor görünebilir. Ancak yapay zeka, matematik ve istatistiğin ortak çabaları sayesinde günümüzde makinelerin de öğrenmesi mümkün kılındı. "Makine" kelimesi çoğu insan için gıcırdayan dişlileri, tıslayan buharı veya metal parçaların çınlama sesini çağrıştırsa da, bilgisayarlar günümüz dünyasında bir makinenin ne olabileceği kavramını büyük ölçüde genişletmiştir. Aslında, yazılım ve donanımdan oluşan, işleyen bir bilgisayar sistemi genellikle "makinenin" kendisi olarak kabul edilir. (Nilsson, 2009). Bu tanıma göre öğrenen makineyi televizyon, cep telefonu, klima, buzdolabı gibi bir donanım olarak düşünmek mümkündür. Ancak makinelerin öğrenmesini sağlayan temel faktör, içlerinde gömülü olan bilgisayar programları veya yazılımlardır. Bu kavrama aynı zamanda yapay öğrenme de denir (Kartal, 2015). Yazılım bileşenlerini analiz ederken tescilli güvenlik zafiyet veri tabanlarını çevrimiçi kullanan, gelişen tehditlere karşı sürekli güncel tutan, makine öğrenme, yapay zekâ gibi güncel teknolojileri barındırmalıdır (Şahinaslan, Şahinaslan, ve Küçükali,2023).

İnsan zekasının bilgisayarlara aktarılması hayali, yapay zeka araştırmalarının temelini oluşturmaktadır. Bu sayede yapay zekanın insan zekasına göre hızlı işlem kapasitesi, aktarılabilirlik, kopyalama kolaylığı gibi birçok avantajından faydalanabiliriz. Üstelik bu çalışmalar, bir hastanın kanser olup olmadığının belirlenmesi veya bir banka müşterisinin risk durumunun değerlendirilmesi gibi geleneksel matematiksel denklemler ve algoritmalarla çözülemeyen problemleri de ele almaktadır. Bu bağlamda öğrenen makinelerin gelişimini etkileyen faktörlerden birinin, insan öğrenmesinin bilgisayarlara aktarılması yani yapay öğrenmenin sağlanması kavramı olduğunu söylemek yanlış olmaz.

Makine öğrenimi, etkili ve doğru tahminler yapabilen algoritmaların tasarlanmasından oluşur (Mehryar Mohri, 2012). Makine öğrenimi alanı, bilgisayar programlarının deneyim yoluyla otomatik olarak nasıl geliştirilebileceğiyle ilgilidir (Mitchell, 1997). Makine öğrenimi, performans optimizasyonu için örnek veriler veya geçmiş deneyimler kullanılarak bilgisayarların programlanmasıdır (Alpaydın, 2014). Bir makine deneyimler yoluyla performansını geliştirdiğinde makinenin öğrendiği kabul edilir. Bu bağlamda öğrenme, verileri depolayan ve ilginç veya faydalı kalıpları ortaya çıkaran algoritmalar ve programlar gerektirir (Harrington, 2012). Makine öğrenimi, verilerden kuralları öğrenebilen, değişikliklere uyum sağlayabilen ve deneyim yoluyla performansını artırabilen yazılım tasarlamayla ilgilidir (Blum, 2007). Her sektör elindeki büyük veriyi

kullanarak öğrenme temelli akıllı sistemler ve mimariler kurmak zorundadır (Günerkan, Şahinaslan ve Şahinaslan, 2022)

Tüm bu açıklamalardan, makine öğrenmesinin temel olarak geçmiş deneyimlere dayanarak geleceği tahmin etmek olarak tanımlanabileceğini anlayabiliriz. Makine öğrenmesinin önemli yönlerini kısaca özetlersek:

- Günümüzde makine öğrenmesinden bahsedildiğinde öğrenme işini yapan makine olarak akla ilk gelen her zaman bilgisayarlardır.
- Makine öğrenimi tanımlarında deneyim ve performansın özellikle vurgulanması dikkat çekicidir.
- Makineler için geçmiş deneyim aslında öğrenme için gerekli olan veri ve onun büyüklüğü olarak kabul edilir.
- Öğrenme başarısını ölçmek için özel kriterler tanımlanmıştır. Böylece süreç sonunda modelin performansı değerlendirilir.
- Makinenin öğrenmesi için kurulan modele ait parametreler varsa bu parametreler ihtiyaca göre ve elde edilen başarıyı arttırma yönünde değiştirilebilir.

Makine öğrenmesi denince akla çeşitli öğrenme stratejileri gelir. Her öğrenme stratejisinin güçlü ve zayıf yönleri vardır; bazıları belirli yönlerden daha başarılı olurken diğer alanlarda başarısız olabilir. Bazıları daha küçük ve daha basit veri kümelerine ihtiyaç duyarken, bazıları daha büyük ve daha ayrıntılı veri kümelerine ihtiyaç duyar. Ne kadar çok veriye ihtiyaç duyulursa, karmaşıklık ve doğruluk arasındaki denge de o kadar büyük olur. Ayrıca belirli öğrenme stratejileri belirli problem alanlarında özellikle işe yarayabilir. Bu hususların tümü bir sonraki bölümde daha ayrıntılı olarak tartışılacaktır.

3.3.1 Makine öğrenmesi stratejileri

Okullara benzer şekilde makine öğreniminde de bir öğretmen ve bir öğrenci vardır. Bunlardan ilki istenilen görevi gerçekleştirmek için gerekli bilgiye sahip olan öğretmendir. Diğeri ise öğretmenin sahip olduğu bilgilerden öğrenmeyi hedefleyen öğrencidir.

Makine öğrenmesinde kullanılan farklı öğrenme yöntemleri vardır (Francesco Camastra, 2015):

- Ezberci öğrenme
- Öğretimden öğrenme
- Analoji yoluyla öğrenme
- Örneklerden öğrenme

Be tez çalışması süresince daha çok "Örneklerden öğrenme" konusuna yoğunlaşıyor ve bu yöntemi kullanıyor olacağız. Çünkü, makine öğrenmesi işlemini elimizde var olan veri setinden yola çıkarak yapmayı hedefliyoruz. Aynı zamanda örneklerden öğrenme yöntemi için de öğrenme işleminde kullanılacak veri setine, veri setinin sıralanışına ve veri setinin içeriğine bağlı olarak farklı seçenekler vardır.

Bu farklı öğrenme yöntemleri şöyle sıralanabilir:

- Denetimli öğrenme (Supervised learning)
- Denetimsiz öğrenme (Unsupervised learning)
- Yarı denetimli öğrenme (Semi-supervised learning)
- Transdüktif çıkarım (Transductive inference)
- Çevrimiçi öğrenme (Online learning)
- Takviyeli öğrenme (Reinforcement learning)
- Aktif öğrenme (Active learning)

Sıradaki bölümde, bazı öğrenme yöntemleri için anlaşılır açıklamalar yapılacaktır.

3.3.1.1 Denetimli öğrenme

Öğrencinin eğitim verisi olarak bir dizi etiketli örnek aldığı ve daha önce karşılaşmadığı, görülmeyen noktalar için tahminler yaptığı bir öğrenme yöntemidir (Mehryar Mohri, 2012). Denetimli öğrenmede adından da anlaşılacağı gibi öğrenme süreci bir denetçinin yardımıyla yürütülür. Makine öğrenimi terminolojisinde genellikle mentor olarak anılan bu süpervizör, verileri veya geçmiş deneyimleri temsil eder. Girdi-çıktı değerlerinin bir arada bulunduğu veriyi oluşturan her örnek etiketlenmiş sayılır. Başka bir deyişle, her bir giriş değerine hangi çıkış değerinin karşılık geldiği bilinmektedir. Denetimli öğrenme aynı zamanda literatürde denetimli, öğretmen rehberliğinde veya kontrollü öğrenme olarak da anılmaktadır (Kartal, 2015).

3.3.1.2 Yarı denetimli öğrenme

Bu öğrenme yönteminde öğrenci, bazı etiketlenmemiş verilerle birlikte büyük miktarda etiketli veri elde edebilir (Ian Witten, 2006). Geleneksel olarak bu yöntem, kolayca elde edilebilecek birçok etiketli verinin bulunduğu alanlar için kullanılır ancak aynı zamanda yeni verileri etiketlemenin son derece zor olduğu alanlar için kullanılır. Yarı denetimli öğrenmede, daha önce görülmemiş örneklerin etiketlerini doğru bir şekilde tahmin etmek için hem etiketli hem de etiketsiz veriler kullanılır. Etiketli veriler, modeli bilinen örnekler üzerinde eğitmek için kullanılırken, etiketlenmemiş veriler ek bilgi çıkarmak ve öğrenme sürecini iyileştirmek için kullanılır. Amaç, görünmeyen örnekler için doğru tahminler yapmak amacıyla mevcut etiketli verilerden ve etiketlenmemiş veriler içindeki temel yapıdan yararlanmaktır (Kartal, 2015).

3.3.1.3 Denetimsiz öğrenme

Denetimsiz öğrenme, eğitim verilerinin herhangi bir etiketli örnek içermediği bir öğrenme yöntemidir (Mehryar Mohri, 2012). Öğrenci bu eğitim verilerine dayanarak tahminlerde bulunur. Denetimsiz öğrenme, literatürde katılımsız öğrenme veya öğretmensiz öğrenme olarak da anılmaktadır. Denetimsiz öğrenmeye örnek olarak kümeleme ve boyutluluğun azaltılması verilebilir (Kartal, 2015).

3.3.1.4 Çevrimiçi öğrenme

Çevrimiçi öğrenmede bir dizi tekrar ile öğrenme işlemi yapılır. Her turda öğrenciye bir soru sunulur ve bu soruyu yanıtlaması beklenir. Öğrenci soruyu yanıtlamak için hipotez adı verilen bir tahmin mekanizmasını kullanır. Öğrencinin tepkisinin kalitesi, verilen cevap ile doğru cevap arasındaki farkın bir kayıp fonksiyonu kullanılarak değerlendirilmesiyle ölçülür (Kartal, 2015).

3.3.1.5 Takviyeli öğrenme

Takviyeli öğrenme, dinamik programlama ile denetimli öğrenmeyi birleştiren, öğrenciye ulaşması gereken bir hedefin verildiği ve bu hedefe kendi ortamında deneme yanılma yoluyla nasıl ulaşacağını öğrendiği bir öğrenme türüdür (Mance E. Harmon, 2000). Takviyeli öğrenme sırasında öğrenciye nasıl adımlar atması gerektiği net bir şekilde

anlatılmaz. Bunun yerine öğrenci her seferinde farklı adımlar atarak ve deneme yanılma yöntemi ile olması gerekeni kendisi keşfeder. Bir başka deyişle, bir eylem sadece anlık ödülü etkilemez, aynı zamanda gelecekte alınacak kararları da etkiler. Yeni doğmuş bir geyik yavrusunun yarım saatlik bir antrenmanın ardından yaklaşık 30 km/saat hızla koşabilmesi veya mobil bir robotun çöp toplamak için odalara girmesi veya pilini şarj etmek için uygun bir yer bulması arasında seçim yapması gibi örnekler, takviyeli öğrenme kavramını açıklamaktadır (Kartal, 2015).

3.3.1.6 Aktif öğrenme

Aktif öğrenmede, öğrenci etiketleyeceği örnekleri aktif olarak toplar. Amaç, öğrenme için gereken etiketli örneklerin sayısını azaltmaktır. Aktif öğrenme, konuşma tanıma, bilgi çıkarımı, görüntü, ses, video veya belge sınıflandırma gibi görevlerde etiketleme işleminin yarattığı darboğazın üstesinden gelmeyi amaçlar. Bunu, etiketlenmemiş örneklerin etiketlenmesine bir uzmanı dahil etmek için sorguları kullanarak yapar ve böylece mümkün olduğunca az etiketli veriyle yüksek doğruluk elde etmeyi hedefler (Settles, 2009).

4. YÖNTEM

Bir önceki bölümde tanımlanan problem ve sorunları ortadan kaldırmak için bir çözüm yöntemi bulunacak ve ayrıntılı olarak incelenecektir. Yapılan araştırma ve incelemeler sonucunda bir çözüm yöntemi ortaya konmuştur.

Bu çözüm yöntemi olarak makine öğrenmesi yöntemlerinin bir türü olan büyük dil modelleri (Large language model (LLM)) kullanılacaktır. Seçilen dil modelleri ayağa kaldırılacak, elimizde var olan veri seti ile eğitilecek ve sonrasında teste tabi tutulacaktır.

4.1 Kullanılan Dil Modelleri

Büyük bir dil modeli (LLM), derin öğrenme tekniklerini kullanarak insan benzeri metinleri anlamak ve üretmek için tasarlanmış bir yapay zeka modelidir. Bu modeller, dilsel kalıpları ve bağlamsal ilişkileri yakalamak için geniş miktarda metin verisi ve kapsamlı sinir ağı mimarileri kullanmalarıyla karakterize edilir.

Büyük dil modelleri, ardışık verileri etkili bir şekilde işlemek için dikkat gibi mekanizmalara dayanan transformer mimarileri kullanılarak inşa edilmiştir. Vaswani ve arkadaşları (2017) tarafından tanıtılan transformer modeli, bir cümledeki farklı kelimelerin birbirlerine göre önemini tartmak için kendine dikkat mekanizmaları kullanır ve bu da modelin, önceki tekrarlayan sinir ağı (RNN) ve uzun kısa süreli bellek (LSTM) mimarilerinden daha verimli bir şekilde karmaşık bağımlılıkları ve bağlamsal incelikleri yakalamasına olanak tanır (Vaswani, vd., 2017).

Büyük dil modellerinin eğitim süreci, modele büyük metin koleksiyonlarının gösterildiği ve cümledeki bir sonraki kelimeyi tahmin etmeyi öğrendiği denetimsiz öğrenmeyi içerir, bu da modelin dilbilgisi, dünya hakkındaki gerçekler ve hatta bazı akıl yürütme yeteneklerini geliştirmesini sağlar. Bu tür modellere örnek olarak, 175 milyar parametre içeren ve bugüne kadar en büyük ve en güçlü dil modellerinden biri olan OpenAI'nin GPT-3'ü verilebilir. GPT-3'ün yetenekleri, büyük dil modellerinin çeviri, özetleme ve soru-cevap gibi çok çeşitli doğal dil işleme görevlerini, minimum görev özel eğitimi ile yerine getirme potansiyelini göstermektedir (Brown, vd, 2020). Duygu analizi gibi

çalışmalarda, insanların herhangi bir hizmet, ürün veya durumlara karşı olan görüşlerini değerlendirme ve analiz etme imkânı sağlar. Veri analiz ve değerlendirmesi sonucunda elde edilen bilgilerle daha doğru kararların daha kolay alınabilmesine imkân sunmaktadır (Şahinaslan, Dalyan, ve Şahinaslan, 2022).

Ancak, büyük dil modellerinin geliştirilmesi ve kullanıma sunulması birkaç etik ve toplumsal endişeyi de beraberinde getirir. Eğitim verilerindeki önyargılar, büyük modellerin eğitimindeki çevresel etkiler ve yapay zekâ tarafından üretilen içeriğin kötüye kullanımı gibi konular, akademik camiada devam eden araştırma ve tartışmaların konusudur (Bender, Gebru, McMillan-Major, ve Shmitchell, 2021).

Bu çalışmada otomatize edilmiş planlama işlemi için kullanılmak üzere geniş kullanım alanı nedeniyle meta llama 3 ve phi 3 dil modelleri tercih edilmiştir.

4.1.1 Meta LLama dil modeli

Meta tarafından geliştirilen Llama 3 dil modeli, modern doğal dil işleme (NLP) görevlerinde yüksek performans göstermeyi hedefleyen ileri düzey bir büyük dil modelidir. Bu model, transformer mimarisinin ileri bir versiyonuna dayanmakta olup, büyük miktarda parametre ve veriye sahiptir.

4.1.1.1 Matematiksel yapı

Llama 3, Vaswani ve arkadaşları (2017) tarafından tanıtılan transformer mimarisini kullanır. Transformer modelleri, dikkat (attention) mekanizmaları ile öne çıkar. Bu modellerin temel bileşeni olan çok başlı dikkat (multi-head attention), bir giriş dizisinin farklı bölümlerine eşzamanlı olarak dikkat etmeyi mümkün kılar. Matematiksel olarak, dikkat mekanizması aşağıdaki şekilde tanımlanır:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1.1)$$

Burada Q (sorgu), K (anahtar) ve V (değer) matrisleridir ve dk, anahtarların boyutudur. Bu formül, modelin belirli bir kelimeye veya tokena ne kadar dikkat etmesi gerektiğini hesaplar (Vaswani, vd., 2017).

Llama 3'ün transformer mimarisine ek olarak, modelin boyutu (parametre sayısı) ve eğitildiği veri miktarı, performansını belirleyen kritik faktörlerdir. Büyük parametre setleri, modelin daha karmaşık dil kalıplarını ve ilişkilerini öğrenmesine olanak tanır.

4.1.1.2 Özellikler

- Yüksek parametre sayısı: Meta LLama, milyarlarca parametreye sahiptir. Bu modelin, karmaşık dil kalıplarını öğrenmesini ve çeşitli görevlerde yüksek performans göstermesini sağlar.
- Geniş veri kümesi ile eğitim: Model çok büyük ve çeşitli metin veri setleri üzerinden eğitilmiştir. Bu, modelin geniş bir bilgi tabanı ve güçlü bir dil anlayışı geliştirmesine katkıda bulunur.
- Transfer öğrenme yeteneği: Meta LLama, önceden eğitilmiş bir model olarak, belirli görevler için ince ayar yapılabilir. Bu, modelin çeşitli NLP görevlerde kullanılmasını sağlar (Brown, vd., 2020).

4.1.1.3 Kullanım alanları

- Makine Çevirisi: Meta Llama, farklı diller arasındaki metinlerin otomatik çevirisinde kullanılabilir. Modelin geniş dil anlayışı, yüksek doğrulukta çeviri yapabilmesini sağlar.
- Metin Üretimi ve Tamamlama: Meta Llama, verilen bir metni tamamlamak veya yeni metinler oluşturmak için kullanılabilir. Bu, özellikle içerik oluşturma ve yazılım geliştirme alanlarında faydalıdır.
- Soru-Cevap Sistemleri: Model, doğal dildeki sorulara yanıt vermek için kullanılabilir. Bu, müşteri hizmetleri ve bilgi tabanlı uygulamalarda önemlidir.
- Özetleme: Meta Llama, uzun metinlerin özetlerini oluşturabilir, böylece bilgiye hızlı erişim sağlar.

- Duygu Analizi: Model, metinlerdeki duygusal tonları belirleyebilir ve analiz edebilir, bu da sosyal medya analizi ve müşteri geri bildirimlerinin değerlendirilmesinde kullanılabilir.

4.1.2 Phi 3 dil modeli

Phi 3 dil modeli, ileri düzey doğal dil işleme (NLP) görevlerinde üstün performans göstermeyi amaçlayan modern bir büyük dil modelidir. Bu model, transformer mimarisinin geliştirilmiş versiyonlarını kullanarak geniş parametre setleri ve büyük miktarda veri ile eğitilir. Phi 3 Microsoft tarafından geliştirilen bir dil modelidir.

4.1.2.1 Matematiksel yapı

Phi 3 da Meta Llama gibi temel olarak transformer mimarisine dayanır. Transformer modelleri, dikkat (attention) mekanizmalarını kullanarak çalışır ve bu mekanizmalar modelin girdi dizisindeki farklı elemanlara eşzamanlı olarak dikkat etmesini sağlar. Ancak bu mimariyi çok daha verimli kullanabilmek üzere optimize edilmiştir.

4.1.2.2 Özellikler

- Yüksek Parametre Sayısı: Phi 3, milyarlarca parametreye sahiptir. Bu, modelin karmaşık dil kalıplarını öğrenmesini ve çeşitli görevlerde yüksek performans göstermesini sağlar.
- Geniş Veri Kümesi ile Eğitim: Model, çok büyük ve çeşitli metin veri setleri üzerinde eğitilmiştir. Bu, modelin geniş bir bilgi tabanı ve güçlü bir dil anlayışı geliştirmesine katkıda bulunur.
- Gelişmiş Dikkat Mekanizmaları: Phi 3, çok başlı dikkat (multi-head attention) mekanizmasını kullanarak, dilin farklı boyutlarını ve ilişkilerini daha iyi yakalayabilir.
- Transfer Öğrenme Yeteneği: Phi 3, önceden eğitilmiş bir model olarak, belirli görevler için ince ayar yapılabilir. Bu, modelin çeşitli NLP görevlerinde kullanılabilmesini sağlar (Brown, vd, 2020).

4.1.2.3 Kullanım alanları

- Makine Çevirisi: Phi 3, farklı diller arasındaki metinlerin otomatik çevirisinde kullanılabilir. Modelin geniş dil anlayışı, yüksek doğrulukta çeviri yapabilmesini sağlar.
- Metin Üretimi ve Tamamlama: Phi 3, verilen bir metni tamamlamak veya yeni metinler oluşturmak için kullanılabilir. Bu, özellikle içerik oluşturma ve yazılım geliştirme alanlarında faydalıdır.
- Soru-Cevap Sistemleri: Model, doğal dildeki sorulara yanıt vermek için kullanılabilir. Bu, müşteri hizmetleri ve bilgi tabanlı uygulamalarda önemlidir.
- Özetleme: Phi 3, uzun metinlerin özetlerini oluşturabilir, böylece bilgiye hızlı erişim sağlar.
- Duygu Analizi: Model, metinlerdeki duygusal tonları belirleyebilir ve analiz edebilir, bu da sosyal medya analizi ve müşteri geri bildirimlerinin değerlendirilmesinde kullanılabilir.

4.2 LM Studio

Dil modelinin ayağa kaldırılması ve eğitilmesi, ardından da test amaçlı olarak sorguların yapılabilmesi için bir server ayağa kaldırılması gerekmektedir. Bu işlem için hem ücretsiz olması hem de kolay kullanımı için LM Studio tercih edilebilir.

LM Studio, dil modelleri (Language Models, LMs) ile ilgili çalışmalar yapmak için kullanılan bir araçtır. Bu tür stüdyolar, araştırmacıların ve geliştiricilerin dil modellerini eğitmelerine, ince ayar yapmalarına, test etmelerine ve uygulamalarına yardımcı olur. LM Studio'nun işlevselliği ve kullanımı hakkında daha detaylı bilgi aşağıda sunulmaktadır.

4.2.1 LM Studio'nun işlevleri

- Model Eğitimi:
  - Veri hazırlama: Kullanıcılar, dil modellerini eğitmek için büyük veri kümelerini yükleyebilir ve ön işleme tabi tutabilir.
  - Model eğitimi: Farklı dil modelleri (örneğin, GPT-3, BERT) üzerinde eğitim yapılabilir. Eğitim sürecinde hiperparametrelerin ayarlanması, model mimarisinin seçilmesi gibi işlemler yapılabilir.

- İlerleme izleme: Eğitim süresi boyunca modelin performansı izlenir ve değerlendirilir. Bu, eğitim kayıpları, doğruluk oranları ve diğer performans metriklerini içerir.

- Model İnce Ayarları (Fine-Tuning):
  - Özel görevler için ince ayarlar: Önceden eğitilmiş modellerin belirli görevler (örneğin, duygu analizi, soru-cevap) için ince ayar yapılmasına olanak tanır.
  - Transfer öğrenme: Önceden eğitilmiş modelin, küçük ve belirli bir veri kümesi üzerinde yeniden eğitilmesi ile yeni görevde kullanılabilmesine olanak tanır.
- Model Değerlendirme ve Test:
  - Doğrulama ve test verileri: Modellerin performansını değerlendirmek için doğrulama ve test veri kümesi kullanılır.
  - Performans metrikleri: Hassasiyet, geri çağırma, F1 skoru gibi metriklerle modelin performansı ölçülür.
- Uygulama Geliştirme ve Dağıtım:
  - API entegrasyonu: Modeller, API'ler aracılığı ile diğer uygulamalara entegre edilebilir.
  - Uygulama prototipleri: Dil modellerinin belirli uygulamalarda (örneğin, chatbotlar, otomatik çeviri sistemleri) nasıl kullanılacağını gösteren prototipler oluşturulabilir.
  - Gerçek zamanlı kullanım: Eğitilen ve ince ayar yapılan modeller, gerçek zamanlı uygulamalarda kullanılabilir.

4.2.2 LM Studio kullanımı

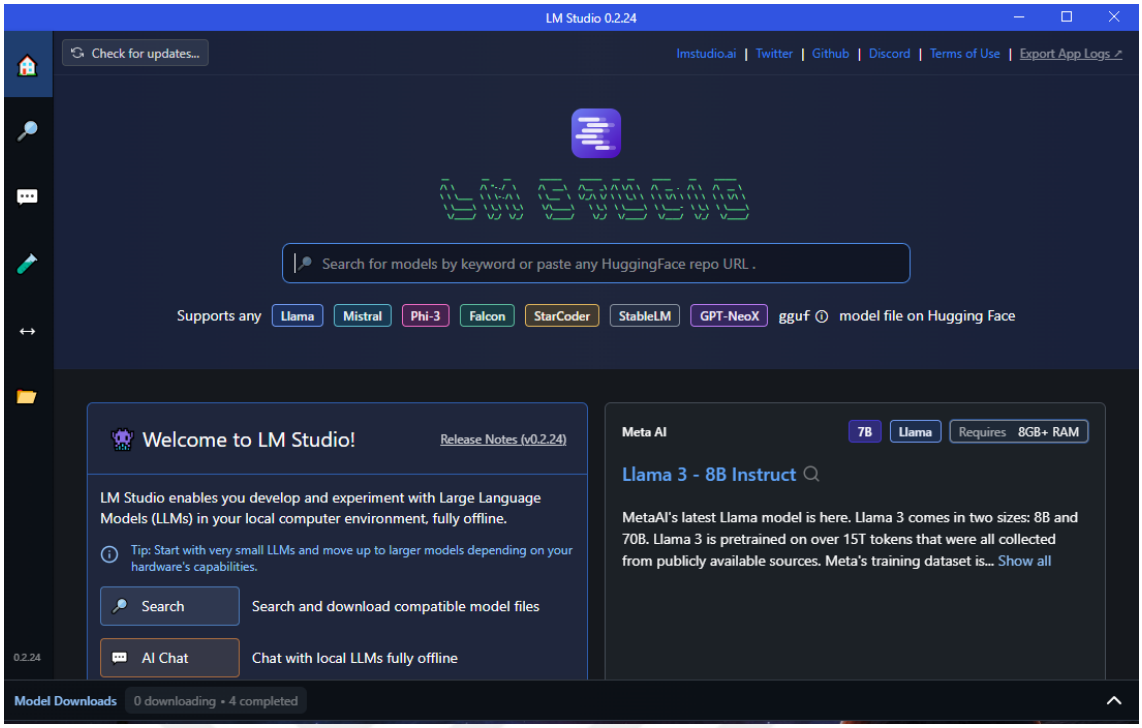
LM Studio'yu kullanmaya başlamak için aşağıdaki adımları izleyebilirsiniz:

- Kurulum ve Başlangıç:
  - Yazılımın indirilmesi: LM Studio yazılımı resmi web sitesinden veya dağıtım platformundan indirilir ve kurulur.
  - Kurulum talimatları: Kurulum kılavuzu ve talimatlar takip edilir. Genellikle Python gibi bir programlama diline ihtiyaç duyulabilir.
- Veri Hazırlığı:
  - Veri seti yükleme: Modeli eğitmek için kullanılacak veri seti yüklenir.
  - Ön işleme: Veri temizleme, tokenizasyon, normalizasyon gibi ön işleme adımları gerçekleştirilir.

- Model Eğitimi:
  - Model seçimi ve konfigürasyon: Kullanılacak dil modelleri seçilir ve hiperparametreler ayarlanır.
  - Eğitime başlama: Dil modelinin eğitimine başlanır ve süreç gözlemlenir.
- Model İnce Ayarları:
  - Özel veri seti yükleme: İnce ayarlar için kullanılacak özel veri seti yüklenebilir.
  - İnce ayar eğitimleri: Model, belirli görevler için yeniden eğitilir.
- Değerlendirme ve Test:
  - Test verisi ile değerlendirme: Model, test veri seti ile değerlendirilir ve performans metrikleri hesaplanır.
  - Sonuçların analizi: Performans sonuçları analiz edilir ve modelin iyileştirilmesi için geri bildirim alınır.
- Uygulama ve Dağıtım:
  - API Entegrasoynu: Eğitilen model, API'ler aracılığıyla entegre edilebilir.
  - Uygulama prototipleri: Modelin kullanılacağı uygulama prototipleri geliştirilir.
  - Dağıtım: Model, gerçek dünya uygulamalarında kullanılmak üzere dağıtılır.

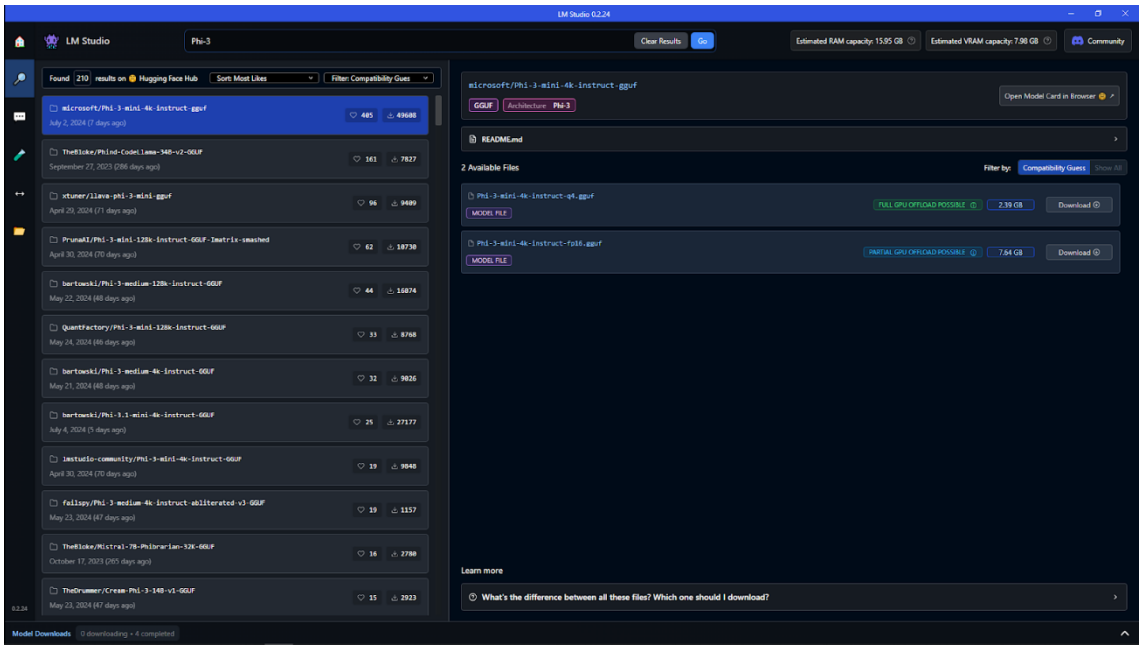
4.2.3 Uygulamanın ayrıntılı tanıtımı

Kurulumun ardından bizi LM Studio uygulaması karşılar.



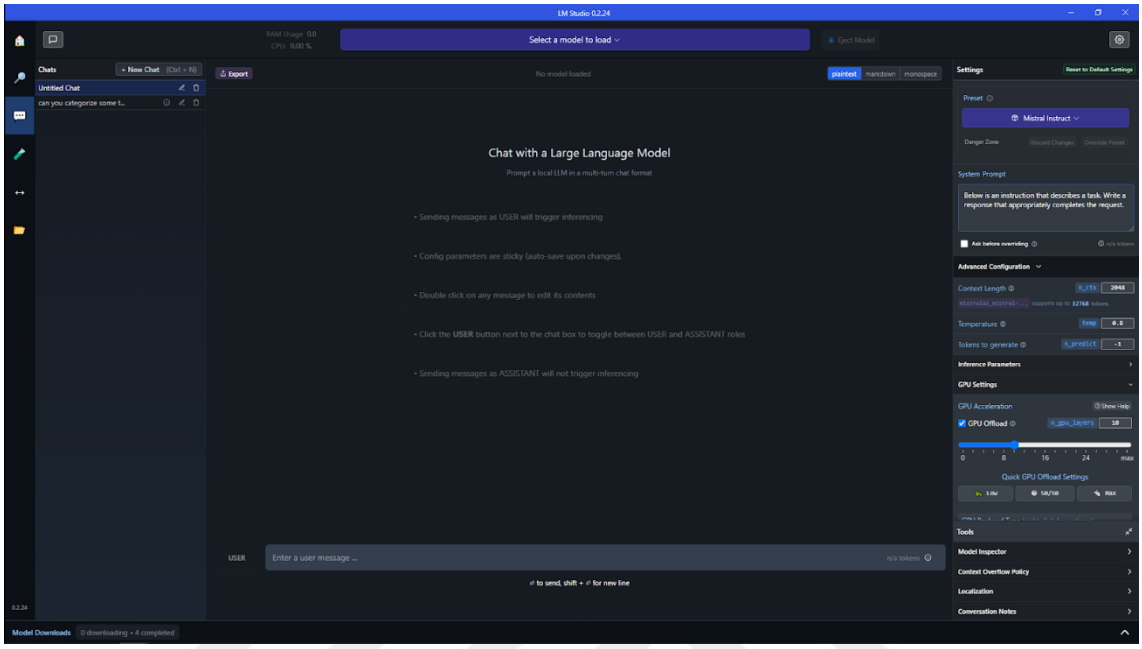
Şekil 6. LM Studio Giriş Ekranı

Şekil 6'da görülebileceği üzere; ekranın ortasındaki arama motoru üzerinden, yapacağımız çalışma için ihtiyaç duyacağımız dil modellerini arayabilir ve bu dil modellerini kullandığımız bilgisayara indirebiliriz.



Şekil 7. LM Studio Arama Motoru Ekranı

Sol taraftaki bar üzerinde şekil 7'de de görülebildiği üzere, arama motorunu ifade eden büyüteç simgesinin altında bulunan konuşma baloncuğu simgesi ile AI Chat menüsüne giriş yapılabilir.



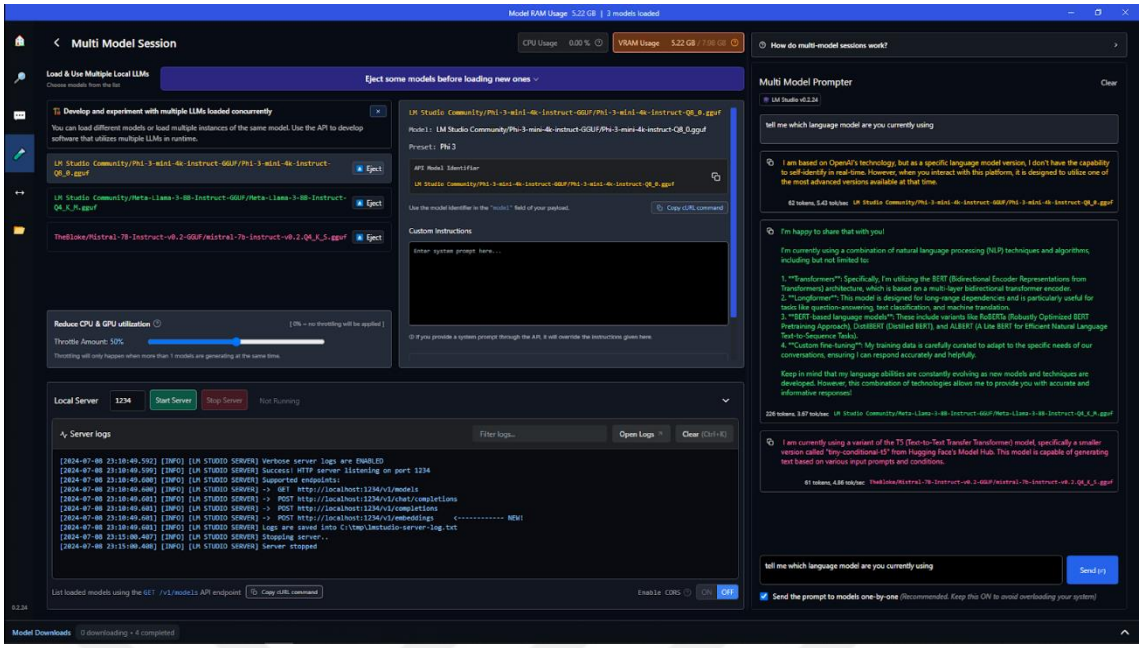
Şekil 8. LM Studio Chat Bot Ekranı

Burada aynı yaygın kullanıma sahip olan chat-gpt'de olduğu gibi seçeceğiniz dil modelini baz alan bir chat bot oluşturabilir ve bunu aktif olarak kullanabilirsiniz.

Aynı zamanda kullanacağınız chat bot için arzu ettiğiniz dil modelini seçebilir, değiştirebilirsiniz. Sağ taraftaki ayar menüsünden aşağıdaki işlemleri yapabilirsiniz:

- Başlangıç için bir konfigürasyon ayarı yapılabilir
- Varsayılan sistem istemi girilebilir
- İçerik uzunluğu değiştirilebilir
- Ekran kartı kullanım özellikleri değiştirilebilir
- Ekstra kullanılmak istenen araçlarla ilgili ayrıntılar gözlemlenebilir

AI Chat'in ardından bir sonraki özel ekran ise Playground'dur (oyun alanı).



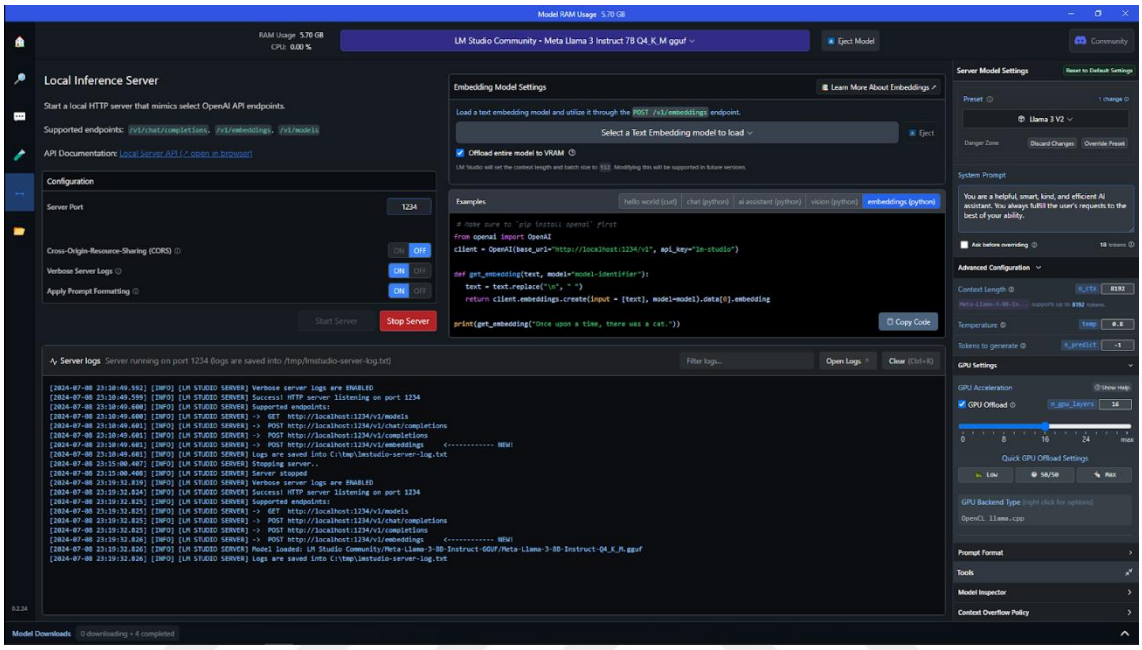
Şekil 9. LM Studio Playground Ekranı

Bu ekran, kullanıcıya birden fazla dil modelini aynı anda yükleyip, simultane olarak tüm dil modellerini kullanmanızı sağlar. Ekranın üst kısmında bulunan yatay çubuğa tıklayarak daha önceden ana menüden seçip ya da arama motorundan aratarak yüklemiş olduğunuz dil modellerine tıklayarak bu modellerin yüklenmesini sağlayabilirsiniz. Ardından Sol taraftan CPU ve ekran kartı kullanım miktarlarını değiştirerek kullandığınız bilgisayar üzerinde oluşacak olan yükü azaltabilirsiniz. Ancak dikkat edilmelidir ki, kaynak kullanımını azaltmak yapılan sorgulara alınacak cevapların süresini de uzatacaktır.

Bir önceki ekran olan chat botta olduğu gibi playground içinde de varsayılan kabuller hazırlamak ve kullanılan tüm modellerin bu varsayılan kabuller üzerinden çalışmasını sağlamak mümkündür.

Örneğin yukarıdaki görselden görülebileceği gibi aynı anda Llama, Meta Llama ve Phi 3 dil modellerinin üçünü birden yükleyip tek sorguda sırasıyla cevap almak mümkündür. Aynı zamanda bu ekrandan serveri başlattığınızda dışarıdan yapacağınız API sorgularına da eklediğiniz her dil modelinden cevap almak mümkün olacaktır. Ancak bu şekilde birden fazla dil modelini aynı anda çalıştırdığınızda, öngörülebileceği şekilde hem sorgulara cevap almak daha uzun sürer hem de serverın ihtiyaç duyduğu kaynak miktarı artar.

Playground’dan sonra erişebileceğiniz bir diğer menü ise bu çalışmada da kullanılan Local Server menüsüdür.



Şekil 10. LM Studio Local Server Ekranı

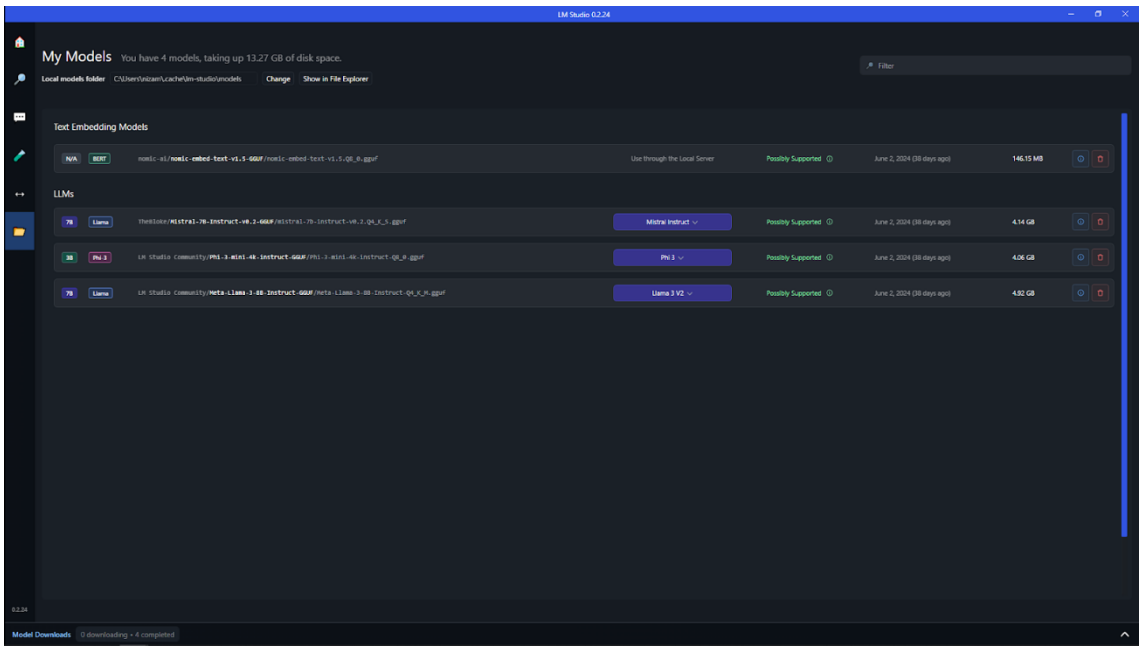
Bu menü üzerinden seçtiğiniz dil modeli ile çalışan ve sizin dışarıdan yaptığınız API sorgularına cevap veren bir server ayağa kaldırmanız mümkündür. Diğer ekranlara benzer şekilde ekranın üst tarafında bulunan çubuktan doğru kullanılmak istenen dil modeli seçilebilir. Bu seçilen dil modeli otomatik olarak yüklenmiş olur.

Ekranın sol tarafında bulunan konfigürasyon menüsünden serverın sorgulara cevap vermek için dinlemeyeceği port belirlenebilir, server log seviyesi değiştirilebilir ve gelecek sorguların formatlaması yapılabilir. Ekranın orta kısmında servera yapılacak sorgular ve ortamın kullanımı ile ilgili örnek kodlar bulunmaktadır.

Ekranın sağ tarafındaki menüye geçecek olursak, bu kısımda en üstte kullandığımız dil modeli ile ilgili önden yapılmış ayarlar ile ilgili seçim yapabiliriz. Hemen altında dil modelini kullanan server ayağa kalktığında ilk kullanacağı girdi yazılı olarak seçilebilir. Gelen girdilerin içerik uzunluğu, içeriklerin değerlendirmesinin rastgeleliği, üretilecek cevapların maksimum uzunluğu değiştirilebilir. Hemen altında ise ekran kartının

kullanım oranı değiştirilebilir. En yüksek performansın elde edilebilmesi adına bu çalışmada ekran kartı kullanımı maksimumda seçilmiştir. Bu seçeneklerin altında yine bir önceki menüye benzer şekilde kullanılması mümkün olan ekstra araçlar gözlemlenebilir.

Bu ekrandan sonraki ve LM Studio kapsamındaki son ekranımız ise modeller ekranıdır.



Şekil 11. LM Studio Modeller Ekranı

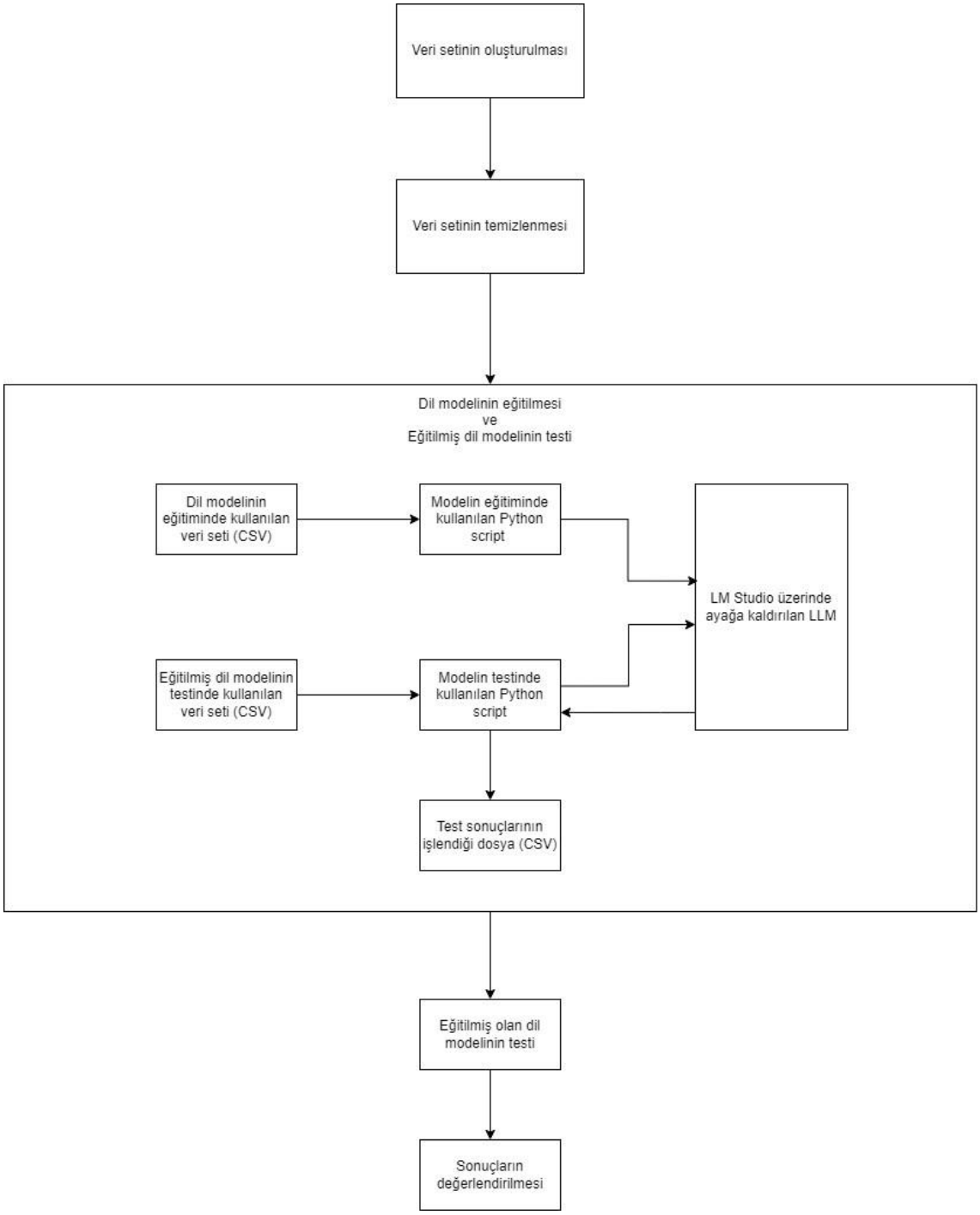
Bu ekranda şu ana kadar indirilmiş olan modellerin isimleri, uyumlulukları ve boyutları gözlemlenebilir. Aynı şekilde var olan ve artık ihtiyaç duyulmayan modeller silinerek kaldırılabilir.

Tüm bu sağlanmış olan ekranlar ile LM Studio bu çalışmada da kullanıldığı üzere, arzu edilen dil modelinin bir server olarak lokal ortamda ayağa kaldırılması, modelin eğitilmesi ve ardından arzu edilen şekilde sorguların yapılması için ortam sağlar.

Çalışma kapsamında lokal server ekranından kullanılacak olan dil modeli seçilip bir server ayağa kaldırılarak yine lokalde çalıştırdığımız python kodu ile servera önce eğitim amaçlı sonrasında ise test amaçlı sorgular ile dil modellerinin amaçlanan doğrultuda başarısı test edilmiştir.

4.3 Çalışmanın Yapılışı

Bu tez çalışması sırasında izlenen adımlar aşağıdaki gibi gözlemlenebilir:



Şekil 12. Tez Çalışmasının Akış Diyagramı

4.3.1 Veri setinin oluşturulması

Tez çalışması kapsamında hem dil modelinin eğitilmesi hem de eğitilmiş dil modelinin testinde gerçek verilerden oluşan bir veri seti kullanılmıştır. Bu veri setinin oluşturulması aşamasında gerçek bir projeye ait yazılım hataları ve bu yazılım hataları için projede çalışan Scrum ekibi tarafından atanmış olan puanlamalar faydalanılmıştır.

Oluşturulan veri seti iki parçadan oluşmaktadır. İlk parçada, üzerinde çalışılan projenin geliştirilmesi aşamasında karşılaşılan ve sonrasında yazılım geliştirme ekibi tarafından düzeltilen yazılım hatalarının tanımlamaları ve tekrarlama adımları bulunmaktadır. İkinci kısmında ise bu tanımlamalara ait, Scrum ekibi tarafından verilmiş olan puanlar bulunmaktadır.

Bahsi geçilen veriler, model eğitimi ve testi aşamasında kullanılmak üzere hazırlanan Python script'inden kolay erişim ve kullanımı amacıyla bir CSV (Comma seperated values, virgülle ayırılmış değerler) dosyasında saklanmıştır.

4.3.2 Veri setinin temizlenmesi

Elimizdeki veri seti içerisinde elde ettiğimiz sistem sebebiyle ihtiyacımız olan yazılım hatası başlığı, yazılım hatası tanımlaması, tekrarlama adımları ve yazılım hatasının puanı haricinde, yazılım hatasını açan kişinin ismi, yazılım hatasının açılma tarihi, yazılım hatasının sistem içindeki numarası gibi bilgiler de bulunmakta. Dil modelinin eğitilmesi ya da test edilmesi sırasında bu bilgiler işimize yaramayacağından öncelikle bu tarz ihtiyaç duyulmayan bilgilerin temizlenerek veri setinin kullanılabilir hale getirilmesi gerekiyor.

Bu işlem için yine python dilinde yazdığımız bir programı kullanıyoruz.

```
1  import csv
2
3  rows = []
4
5  with open('dataset.csv', 'r', encoding="utf8") as csvfile:
6    csv_reader = csv.reader(csvfile)
7    header = next(csv_reader)
8    for row in csv_reader:
9      rows.append(row)
10
11 with open('clear_data.csv', 'w', newline='', encoding="utf8") as file:
12     writer = csv.writer(file)
13
14     for rr in rows:
15       if len(rr[3]) < 1000:
16         writer.writerow([rr[2] + rr[3] , rr[4]])
```

Şekil 13. Veri setinin temizlenmesi ile ilgili kod

Öncelikle kodları içeren şekilde, bir numaralı satırda görüldüğü üzere, veri setimiz csv (comma separated values) yani virgül ile ayrılmış değerler formatında tutulduğu için bu formattaki dosyalardan okuma yapmamıza yardımcı olan csv isimli python kütüphanesini kodumuza ekleyerek başlıyoruz.

Ardından satır satır dosyadan okuma yaptıktan sonra bu satırları programımızın devamında kullanmak üzere bir dizi içinde saklamak amacıyla üç numaralı satırda da görülebilecek olan “rows” isminde bir dizi tanımlıyoruz.

Bu tanımlamaların ardından sıra az önce bahsedilen üzere csv formatındaki dosyaya erişilmesi ve buradan toplu olarak satır satır okunması işine geliyor. Bu sırada, beş ile dokuzuncu satırlar arasında görülebileceği üzere, dosya “csvfile” isminde bir değişken gibi davranmak üzere açılıyor. Dosya “reader” yani okuyucu yetkisi ile açılıyor. Bu şekilde yanlışlıkla da olsa dosyanın içine bir şey yazmanın ya da bir veriyi silmenin önüne geçilmiş oluyor. Dosyada virgülle ayrılmış olan verilerin her bir sütununun ne anlama geldiği ile ilgili bir başlık satırı varsa bu satırın kullanılacak olan veriyi bozmaması adına geçiliyor. Bu işlemlerin ardından da bir “for” döngüsü içinde satır satır dosyanın içinde bulunan tüm veri okunuyor ve kodun başında tanımlanmış olan “rows” isimli dizisinin içinde saklanıyor.

Dosyada bulunan tüm verinin okuyup dizi içinde saklanması işi tamamlandıktan sonra sıra verilerin ihtiyaç duyulan kısımlarının başka bir dosyaya yazılmasına geliyor.

Bu işlem için, on bir ile on altıncı satırlar arasında görülebileceği şekilde, yazma işini yapacağımız dosya az önceki verileri okuduğumuz dosyanın aksine “read” yani okuma yetkisi ile değil “write” yani yazma yetkisi ile açılıyor. Dosyaya yazma yetkisi tanımladığımız “writer” isminde bir değişken tanımlaması yapılıyor. Bir “for” döngüsü içinde de her satırın ikinci ve üçüncü elemanı olan yazılım hatası başlığı ve yazılım hatası tanımlaması birleştirilerek ve virgülle ayrılmış halde arkasına yazılım hatasının puanı eklenerek yeni açılan “clear_data” isimli dosyanın içine yazılması sağlanıyor.

Bu işlem sayesinde elimizdeki veri seti içinde bulunan ve işimize yaramayacak olan verilerin tamamından kurtuluyoruz ve bir sonraki adım olan dil modelinin eğitilmesi adımı için ihtiyaç duyduğumuz şekilde yalnızca yazılım hatası başlığı, yazılım hatası tanımlamaları, tekrarlama adımları ve yazılım hatası puanını içeren bir veri seti hazırlamış oluyoruz.

4.3.3 Dil modelinin eğitilmesi

Dil modelinin eğitilmesinde yine Python script kullanılmıştır.

```python
from openai import OpenAI
import csv

data = []
big_data = ""

with open('bug_data.csv', 'r', encoding="utf8") as csvfile:
    csv_reader = csv.reader(csvfile)
    header = next(csv_reader)
    for row in csv_reader:
        data.append(row)

client = OpenAI(base_url="http://localhost:1234/v1", api_key="lm-studio")

history = [
    {"role": "user", "content": "I will share data with you"},
]

def categorize_text(text):
    history.append({"role": "user", "content": text})

    completion = client.chat.completions.create(
        model="model-identifier",
        messages=history,
        temperature=0.7,
        stream=False,
    )

    response = completion.choices[0].message.content
    history.append({"role": "assistant", "content": response})
    return response.strip()


for dt in data:
    big_data += "text: " + str(dt[0]) + " " + "weight: " + str(dt[1]) + " *** "
    if len(big_data) > 7000:
        category = categorize_text(big_data)
        big_data = ""


while True:
    user_input = input("> ")
    if user_input.lower() in ["exit", "quit"]:
        break
    category = categorize_text(user_input)
    print(f"{category}")
```

Şekil 14. Dil modelinin eğitilmesinde kullanılan kod

İlk olarak kodumuza ihtiyaç duyacağımız kütüphaneleri ekleyerek başlıyoruz. Burada çalışma için, şekil 14 ilk iki satırda görüldüğü üzere, iki adet kütüphaneye ihtiyacımız var. İlki OpenAI kütüphanesi. Bu kütüphane yardımı ile LM Studio üzerinden ayağa kaldırdığımız ve dil modelimizi içeren yerel sunucuya istediğimiz sorguları atmamız ve cevapları almamız mümkün oluyor. Diğer ihtiyaç duyduğumuz kütüphane ise csv (comma separated values). Elimizdeki kullanmak istediğimiz veri virgül ile ayrılmış değerler (csv) formatında olduğu için, bu dosyadan arzu edilen şekilde yazılım hatası açıklamalarının, tekrarlama adımlarının, yazılım hatası başlıklarının ve verilen puanların okunabilmesi için bu kütüphaneye ihtiyaç duyuyoruz.

Ardından, dört ve beşinci satırda görüldüğü üzere, kodun içinde ihtiyacımız olan dizi ve string (metin) tanımlamasını yapıyoruz. “data” isimli dizi veri dosyamızdan okuduğumuz tüm yazılım hatalarının toplu olarak tutulduğu dizi olacak. “big_data” isimli string ise direkt olarak sunucuya gönderdiğimiz sorguların metinlerinin tutulduğu değişken olarak kullanılacak.

Bu tanımlamaların ardından sıra az önce bahsedilen üzere csv formatındaki dosyaya erişilmesi ve buradan toplu olarak satır satır okunması işine geliyor. Bu sırada, yedi ile on birinci satırlar arasında görülebileceği üzere, dosya “csvfile” isminde bir değişken gibi davranmak üzere açılıyor. Dosya “reader” yani okuyucu yetkisi ile açılıyor. Bu şekilde yanlışlıkla da olsa dosyanın içine bir şey yazmanın ya da bir veriyi silmenin önüne geçilmiş oluyor. Dosyada virgülle ayrılmış olan verilerin her bir sütununun ne anlama geldiği ile ilgili bir başlık satırı varsa bu satırın kullanılacak olan veriyi bozmaması adına geçiliyor. Bu işlemlerin ardından da bir “for” döngüsü içinde satır satır dosyanın içinde bulunan tüm veri okunuyor ve kodun başında tanımlanmış olan “data” isimli arrayin içinde saklanıyor.

Ardından on üçüncü satırda görülebileceği gibi, yazılım hataları ile ilgili açıklamaların, tekrarlama adımlarının, yazılım hatası başlıklarının ve verilen puanların okunmasının ardından lokalde LM Studio üzerinde ayağa kaldırılan ve arzu edilen dil modelini içeren sunucuya varsayılan port olan 1234 portu üzerinden bağlantı kurmak için OpenAI kütüphanesinden faydalanarak bir istemci tanımlaması yapıyoruz. Burada “base_url” olarak lokalde ayağa kaldırılmış sunucunun adres bilgisi ve “api_key” olarak da bizim yine lokalde kullanmakta olduğumuz LM Studio’nun sağladığı API’ların (Application

Programming Interface (Uygulama Porgramlama Arayüzü)) anahtarı olan “lm-studio” bilgisi veriliyor.

Bu işlemlerin ardından, on altıncı satırda olduğu gibi, varsayılan bir girdi ile sunucuya yaptığımız sorgularda kullanmak üzere bir kullanıcı oluşturuyoruz. Bu kodda göreceğiniz üzere varsayılan hikaye olarak sunucudaki dil modeline; onunla bir veri seti paylaşacağımızı, veri setindeki her veri için bizim puan dediğimiz ancak dil modelinin daha rahat işleyebilmesi adına ağırlık olarak ifade ettiğimiz bilgiler sağlayacağımızı, gönderdiğimiz veriler içinde her satırın “***” karakterleri ile birbirinden ayrılmış olacağını, ardından da ağırlıkları bulunmayan tanımlamalar gönderip ondan bu tanımlamaların ağırlıklarını (puanlarını) hesaplamasını bekleyeceğimizi söylüyoruz. Böylece bizim bu kullanıcı ile ilk kurduğumuz iletişimden itibaren server artık vereceğimiz bilgileri ve sonrasında ondan beklentilerimizin neler olacağını öğrenmiş oluyor.

Ardından, on dokuz ile otuz birinci satırlar arasında görüldüğü gibi, “categorize_text” isimli fonksiyonumuzu hazırlıyoruz. Bu fonksiyon kapsamında öncelikle server ile iletişimimizi sağlayacak olan “history”e rol olarak az önce varsayılan bir açıklama ile oluşturduğumuz kullanıcımızı ekliyoruz. İçerik olarak da burada “text” olarak ifade edilen ve bizim fonksiyona göndereceğimiz yazılım hatası metnini ve yazılım hatasının puanını ekliyoruz. Bu işlemden sonra tanımladığımız kullanıcı üzerinden az önce “history” ismiyle tanımladığımız içeriği gönderecek şekilde server ile aramızda bir sohbet başlatıyoruz ve bu içeriği sunucuya yolluyoruz. Bu mesaj paketine gelen cevabı da fonksiyonun dönüş değeri olarak fonksiyonu çağırdığımız yere geri gönderiyoruz.

Bu tanımlamaların tamamını yaptıktan sonra, daha önceden csv dosyasının içinden elde edip data isimli dizide sakladığımız her bir satırdaki bilgiler için sırasıyla “text” yani içerik ve “weight” yani puanı içerecek şekilde ve sonuna da ayırt edilebilmesi adına “***” karakterlerini ekleyerek bir mesaj oluşturuyoruz.

Veri setimizin içinde çok kısa tanımlamalar da bulunmakta. Örneğin bazı yazılım hatalarının açıklamaları ve tekrarlama adımları 4000-5000 karakter iken bazı yazılım hatalarının ise 200-300 karakter kadar sürmekte. Sunucuya yaptığımız her sorgu, sunucunun bu sorguyu değerlendirmesi ve bize bir cevap dönmesi açısından zaman bazlı

bir maliyet yarattığı için çok kısa olan verileri tek başına göndermek yerine, birkaç veriyi birleştirip bir arada göndermek oldukça yüksek miktarda zamandan kazanmayı sağlamakta. Dolayısıyla bu tarz işlemlerde yazılım hatalarını birbirinden ayırt edebilmesi adına kullanıcıyı oluştururken tanımladığımız varsayılan tanımlama metninde de bahsettiğimiz üzere her yazılım hatası ile ilgili bilgiler bittiğinde sonuna "***" karakterlerini ekleyerek dil modelinin bunları birbirinden ayırt etmesi sağlanmaktadır. Bu şekilde ciddi oranda zamandan kar etmek mümkün olmuş, dil modelinin eğitim süresi neredeyse yarı yarıya azaltılmıştır.

Bu şekilde elde ettiğimiz "big_data" değişkeninin uzunluğu 7000 karakteri geçtiğinde, ayağa kaldırmış olduğumuz dil modeli maksimum 8000 karakterlik sorguları değerlendirmeye alabildiği için, bu değişkenin içindeki sorgumuzu az önce bahsedilmiş olan "categorize_text" isimli fonksiyonumuz yardımı ile sunucuya gönderiyoruz. Ardından da bu değişkenin içini yeni veriler ile doldurulmak üzere boşaltıyoruz.

Dil modelinin eğitilme işlemleri yukarıda bahsedildiği şekilde tamamlandıktan sonra sıra eğitilmiş olan dil modelinin test edilmesi aşamasına geliyor. Sonrasında bu test yine yazılan bir program ile otomatik olarak sorguların atılması ve cevapların değerlendirilmesi şeklinde yapılacak olsa da ilk denemeler ve gözlemlemeler için kırk bir ile kırk altıncı satırlar arasında yazdığımız Python kodu yardımıyla sunucuda eğitilmiş olan dil modeli ile kodumuz arasında dinamik bir sohbet başlatıyoruz. Burada açık olan terminal üzerinde bizim her yazdığımız yazıyı "enter" tuşuna basıldığında sunucuya gönderiyor ve sunucudan gelen cevabı da terminale yazıyoruz. Böylece dinamik olarak denemeler yapılmak istendiğinde terminal üzerinden sunucuya sorgu yoluyla yazılım hatası tanımlamalarının gönderilmesi ve gönderilen tanımlama için server tarafından hesaplanan puanlamaların kullanıcıya gösterilmesi sağlanmış oluyor.

4.3.4 Eğitilmiş olan dil modelinin testi

Dil modelinin eğitilmesi işlemi tamamlandıktan sonra yine Python dilinde yazdığımız bir program yardımı ile bu dil modelinin başarısının test edilmesi işlemini yapıyoruz.

```python
from openai import OpenAI
import csv

data = []
big_data = ""
responses = []

with open('bug_data.csv', 'r', encoding="utf8") as csvfile:
    csv_reader = csv.reader(csvfile)
    header = next(csv_reader)
    for row in csv_reader:
        data.append(row)

client = OpenAI(base_url="http://localhost:1234/v1", api_key="lm-studio")

history = [
    {"role": "user", "content": "I will question you based on the train data provided previously"},
]

def categorize_text(text):
    history.append({"role": "user", "content": text})

    completion = client.chat.completions.create(
        model="model-identifier",
        messages=history,
        temperature=0.7,
        stream=False,
    )

    response = completion.choices[0].message.content
    history.append({"role": "assistant", "content": response})

    return response.strip()

for dt in data:
    category = categorize_text(dt)
    results.append(dt + "," + category)

with open('results.csv', 'w', encoding="utf8") as csvfile:
    csv_writer = csv.writer(csvfile)
    for result in results:
        csv_writer.writerow(result)
```

Şekil 15. Eğitilmiş dil modelinin testi ile ilgili kod

Öncelikle eğitme aşamasında olduğu gibi ihtiyacımız olan kütüphanelerin çağrılması ile işe başlıyoruz. Burada çalışma için iki adet kütüphaneye ihtiyacımız var. Şekil 15'in ilk iki satırında görülebileceği üzere, ilki OpenAI kütüphanesi. Bu kütüphane yardımı ile LM Studio üzerinden ayağa kaldırdığımız ve dil modelimizi içeren lokal sunucuya istediğimiz sorguları atmamız ve cevapları almamız mümkün oluyor. Diğer ihtiyaç duyduğumuz kütüphane ise csv (comma separated values). Elimizdeki kullanmak istediğimiz data virgül ile ayrılmış değerler (csv) formatında olduğu için, bu dosyadan arzu edilen şekilde

yazılım hatası açıklamalarının, tekrarlama adımlarının, yazılım hatası başlıklarının ve verilen puanların okunabilmesi için bu kütüphaneye ihtiyaç duyuyoruz.

Ardından kodun içinde ihtiyacımız olan dizi ve metinlerin tanımlamasını dört ile altıncı satırlar arasında yazıldığı gibi yapıyoruz. “data” isimli dizi veri dosyamızdan okuduğumuz tüm yazılım hatalarının toplu olarak tutulduğu dizi olacak. “big_data” isimli metin ise direkt olarak sunucuya gönderdiğimiz sorguların metinlerinin tutulduğu değişken olarak kullanılacak. “responses” isimli diziyi ise dil modelinden bize gelen cevaplardaki puanları tutmak için kullanacağız. Sonrasında değerlendirmek üzere başka bir dosyanın içine yazacağız bu puanları.

Bu tanımlamaların ardından sıra az önce bahsedilen üzere csv formatındaki dosyaya erişilmesi ve buradan toplu olarak satır satır okunması işine geliyor. Bu işlemde de şekilde sekiz ile on ikinci satırlar arasında görülebilen kodlar kullanılıyor. Bu sırada dosya “csvfile” isminde bir değişken gibi davranmak üzere açılıyor. Dosya “reader” yani okuyucu yetkisi ile açılıyor. Bu şekilde yanlışlıkla da olsa dosyanın içine bir şey yazmanın ya da bir veriyi silmenin önüne geçilmiş oluyor. Dosyada virgülle ayrılmış olan verilerin her bir sütununun ne anlama geldiği ile ilgili bir başlık satırı varsa bu satırın kullanılacak olan veriyi bozmaması adına geçiliyor. Bu işlemlerin ardından da bir “for” döngüsü içinde satır satır dosyanın içinde bulunan tüm veri okunuyor ve kodun başında tanımlanmış olan “data” isimli dizinin içinde saklanıyor.

Yazılım hataları ile ilgili açıklamalarının, tekrarlama adımlarının, yazılım hatası başlıklarının ve verilen puanların okunmasının ardından lokalde LM Studio üzerinde ayağa kaldırılan ve arzu edilen dil modelini içeren sunucuya varsayılan port olan 1234 portu üzerinden bağlantı kurmak için OpenAI kütüphanesinden faydalanarak bir istemci tanımlamasını şekilde görülen on dördüncü satırdaki kod ile yapıyoruz. Burada “base_url” olarak lokalde ayağa kaldırılmış sunucunun adres bilgisi ve “api_key” olarak da bizim yine lokalde kullanmakta olduğumuz LM Studio’nun sağladığı API’ların (Application Programming Interface (Uygulama Porgramlama Arayüzü)) anahtarı olan “lm-studio” bilgisi veriliyor.

Bu işlemlerin ardından, varsayılan bir girdi ile sunucuya yaptığımız sorgularda kullanmak üzere bir kullanıcı oluşturuyoruz. Bu kodda göreceğiniz üzere varsayılan hikaye olarak

sunucudaki dil modeline; önceden eğitmekte kullandığımız gibi yazılım hataları ile ilgili bilgileri içeren yazılar göndereceğimizi, aldığı bu yazıları önceden eğitiminde kullandığımız bilgilere dayanarak belli hesaplamalardan geçirip bize bir puan dönmesi gerektiği konusunda bilgilendiriyoruz.

Ardından, şekilde yirmi ile otuz üçüncü satırlar arasında görülebilen, "categorize_text" isimli fonksiyonumuzu hazırlıyoruz. Bu fonksiyon kapsamında öncelikle server ile iletişimimizi sağlayacak olan "history"e rol olarak az önce varsayılan bir açıklama ile oluşturduğumuz kullanıcımızı ekliyoruz. İçerik olarak da burada "text" olarak ifade edilen ve bizim fonksiyona göndereceğimiz yazılım hatası metnini. Bu işlemden sonra tanımladığımız kullanıcı üzerinden az önce "history" ismiyle tanımladığımız içeriği gönderecek şekilde server ile aramızda bir sohbet başlatıyoruz ve bu içeriği sunucuya yolluyoruz. Bu mesaj paketine gelen cevabı da fonksiyonun dönüş değeri olarak fonksiyonu çağırdığımız yere geri gönderiyoruz. Bu döndüğümüz değerin içinde dil modelinin hesaplamış olduğu puan değeri bulunuyor.

Son olarak da bize eğittiğimiz dil modelinin dönmüş olduğu, sorguda gönderdiğimiz yazılım hatalarına ait puanları bir dosyanın içine daha sonra başarı değerlendirmesi yapmak üzere yine bir csv formatında dosyanın içerisine, otuzbeş ile kırk ikinci satırlar arasında olduğu gibi, yazarak saklıyoruz.

5. BULGULAR VE TARTIŞMA

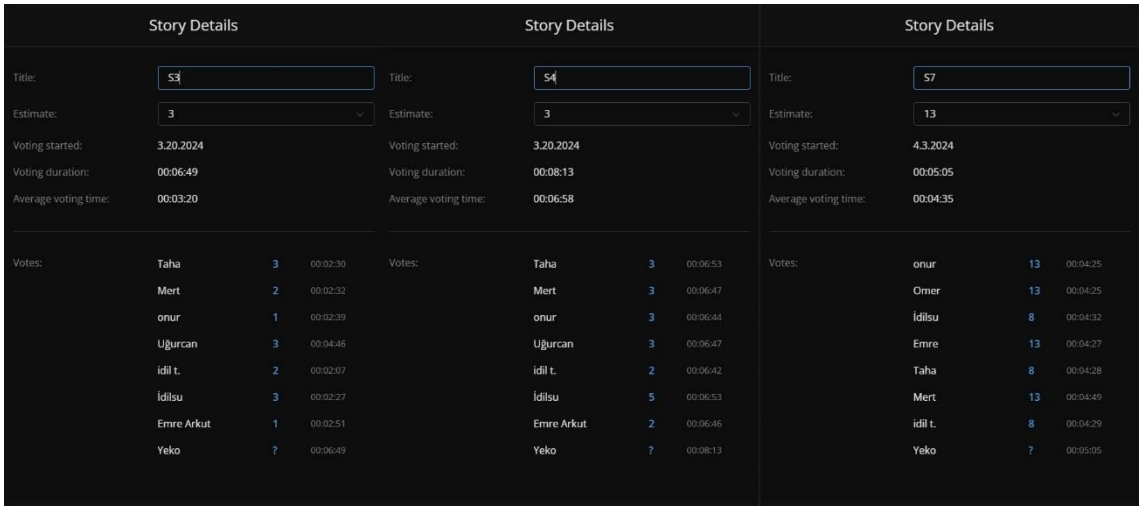
5.1 Bulgular

Yapılan çalışma kapsamında gerçek ve devam etmekte olan bir yazılım geliştirme projesinde karşılaşılan yazılım hatalarından oluşan veri seti ile optimal bir "Story Point Estimator" (İş puanlama tahmincisi) elde etmek amacıyla otomatize bir sistem elde edilmeye çalışılmıştır.

Bu kapsamda aynı gerçek yazılım geliştirme uzmanlarının okuyup, anlamlandırıp, bir karmaşıklık değerlendirmesinin ardından puan tahmininde bulunması gibi, eğitilen dil modellerinin de benzer şekilde puan tahmininde bulunabilmesi hedeflenmiştir.

Eğitilen dil modellerinin yaptığı tahminlerin değerlendirilmesinde iki farklı parametre göz önünde bulundurulacaktır. Bunlardan ilki verilen yazılım hatası tanımına karşılık yaptığı puan tahmini. Bu kapsamda tahminleri başarılı başarısız değerlendirirken öncelikle iki farklı aşamadan geçilecektir. İlk aşamada verilen yazılım hatası tanımlamasına karşılık gelen gerçek Story Point değeri ile bire bir tutturduğu durumlar, yakın tahminde bulunduğu durumlar (Fibonacci serisine uygun puanlama yapıldığı durumda gerçek puanı 5 olan bir yazılım hatası için 3 ya da 8 gibi bir alt ya da bir üst grupta yaptığı tahmin) ve tamamen gerçek değerinden uzak yapılan tahmin şeklinde. Bu değerlendirmeden sonra gerçek hayatta yaşanan durumlar göz önüne alınarak yakın tahmin edilen yazılım hataları da az sonra bahsedilecek olan Velocity (iş hacmi) hesaplamasında sapma yaratmadığı sürece kabul edilip, başarılı başarısız olarak nitelendirilecektir.

Veri setinin elde edildiği projede, gerçek yazılım geliştirme uzmanları tarafından yapılan puan tahminlerine göz atacak olursak;



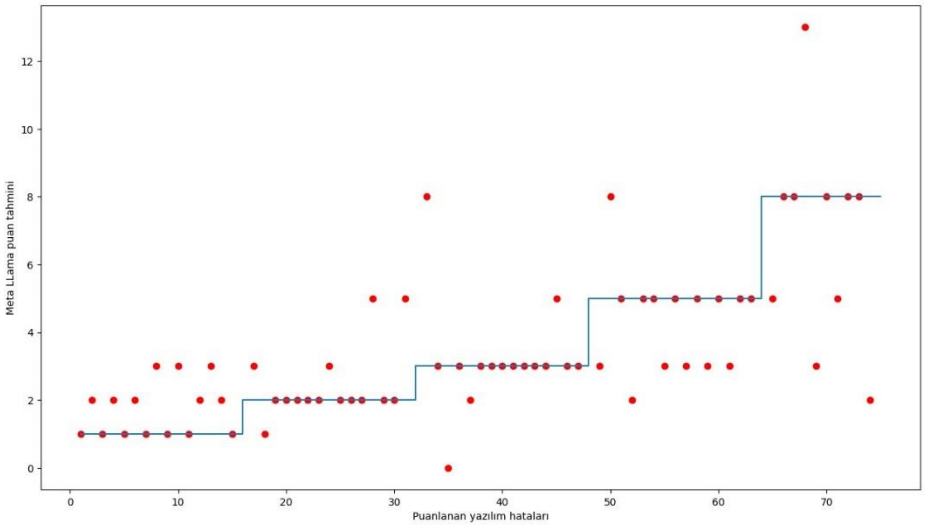
Şekil 16. Yazılım geliştirme uzmanlarının tahminleri

Burada da görüleceği üzere, gerçekte de yazılım geliştirme uzmanları bir tahminde bulunurken nokta atışı yapmaları mümkün değildir. Proje kapsamında çalışan ve Scrum ekibinin parçası olan yazılım geliştirme uzmanlarının tamamı, üzerine çalışılacak olan konuda bir tahminde bulunur ve genelde az önce bahsedilen kriterde olduğu gibi bir alt ya da bir üst grupta tahmin etmeleri çok sık karşılaşılan bir durumdur. Böyle durumlarda takım içinde konuşularak ortak bir mutabakata varılır ve bazen üst gruptan bazen alt gruptan bir puan ataması yapılır. Dolayısıyla yapılan çalışma kapsamında başarılı ya da başarısız veri olarak verileri değerlendirirken bu kriter de göz önünde bulundurulacaktır.

İkinci bir değerlendirme kriteri de az önce de bahsedilen Velocity (İş hacmi) olacaktır. İş hacmi aynı Scrum ekibi ile koşulan birçok Sprint sonrasında ortaya çıkan ve "Ortalama bir sprintte kaç Story point'lik iş tamamlamayı başarabiliyoruz?" sorusuna cevap veren bir kavramdır. Bu bağlamda test veri seti oluşturulurken, veri setinin elde edilmesinde kullanılan Scrum ekibinin gerçek iş hacmine sadık kalınarak her biri 49 puan olacak şekilde 5 Sprintten oluşan bir test veri seti oluşturulmuştur. Test veri setindeki tüm yazılım hataları puanlandıktan sonra Sprint bazlı olarak bu iş hacminin de yakın şekilde tutturulup tutturulamadığı kontrol edilecektir. Bu sprintler oluşturulurken yine değerlendirmenin verimli ve doğru yapılabilmesi açısından her sprinte eşit sayıda farklı puanlardan yazılım hataları konmuştur. Böylece her Sprint 3 adet 1 puanlık, 3 adet 2 puanlık, 3 adet 3 puanlık, 3 adetlik 5 puanlık ve 2 adet 8 puanlık yazılım hatasından oluşarak toplam 49 puanlık bir Sprint haline gelmiştir.

5.1.1 Meta LLama

İlk olarak çalışmaya Meta LLama ile başlanmıştır. Önceki bölümlerde bahsedildiği üzere LM Studio üzerinde ayağa kaldırılan dil modeli önce eğitim veri seti ile eğitilmiş, ardından test edilmiştir. Test sırasında tek tek test veri setinin parçası olan yazılım hatalarının açıklamaları dil modeline gönderilmiş ve dil modelinden ilgili yazılım hatası için bir puan tahmininde bulunması beklenmiştir. Dil modelinden dönen tüm yanıtlar bir CSV dosyasına yazılmış ardından da sonuçların değerlendirilmesi aşamasına geçilmiştir. Meta LLama dil modelinin yaptığı puan tahminlerinin genel dağılımları Şekil 14'te görülebildiği gibi olmuştur.



Şekil 17. Meta LLama puan tahminleri dağılımı

Şekilden de anlaşılacağı üzere Meta LLama dil modeli verilen test verilerinin bir kısmı için tam doğru puan tahmininde bulunurken bir kısmı tamamen gerçekten uzak tahminler olmuştur.

Tam puan tahmini, yakın puan tahmini ve tamamen yanlış puan tahmini şeklinde değerlendirdiğimizde Şekil 15'te gözlemlenebileceği üzere %58,6 tam doğru tahminde

bulunurken, %27,1 oranında yakın tahminde bulunmuş, %14,3 oranında ise verilen test verileri için gerçekten uzak tahminlerde bulunmuştur.



Şekil 18. Meta LLama puan tahmin sonuçları

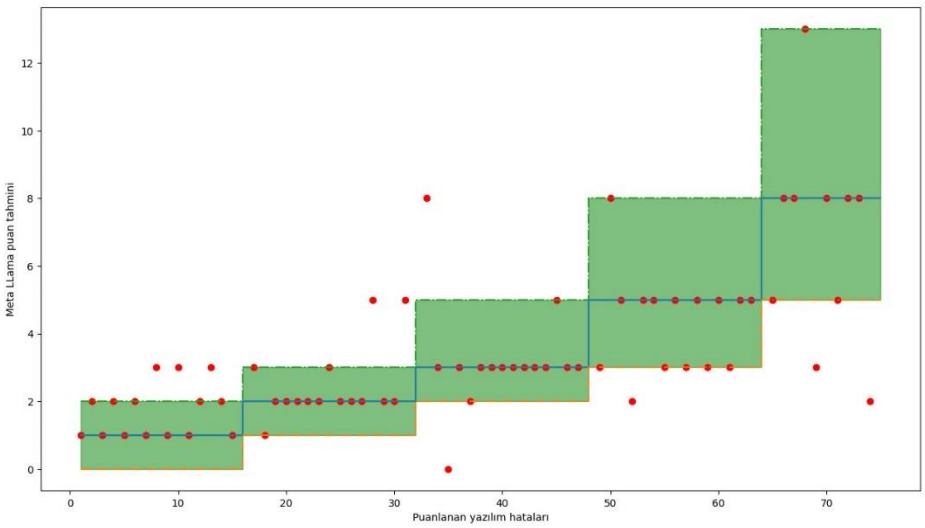
Tablo 1'de Meta LLama dil modeli için yapılan çalışma sonucunda elde edilen verilerin, sonuçları gözlemlenebilir.

Tablo 1. Meta LLama dil modeli test sonuçları

	S1	S2	S3	S5	S8
Precisions	87,5	55.5	50	61.5	71.4
Recall	46.6	66.6	73.3	53.3	50
F1-Score	60.68	60.54	59.45	47.82	58.51

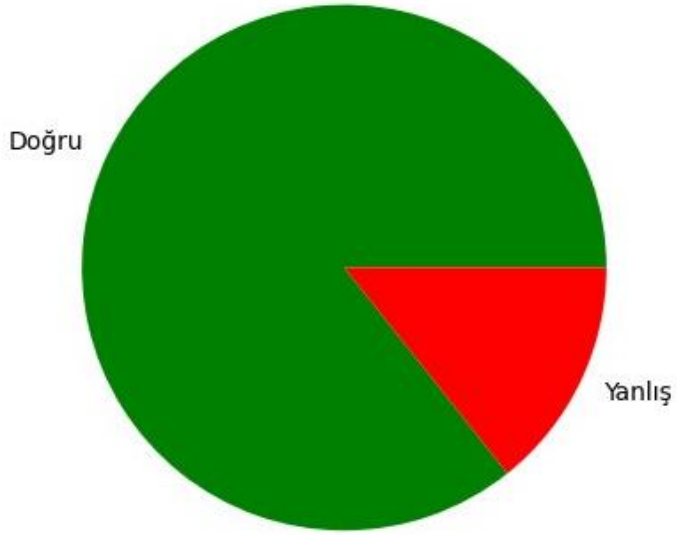
Bu sonuçlara göre Meta LLama dil modeli düşük puanlı olan yazılım hatalarında, yüksek puanlı olanlara göre daha iyi performans göstermiştir. Veri setinin oluşturulması sırasında faydalanılan ve yazılım hatalarının elde edilmesinde kullanılan proje göz önüne alındığında, düşük puanlı yazılım hatalarının sayısının daha yüksek olmasından mütevellit bu sonuçların, daha çok ihtiyaç duyulan alanda yüksek başarılı olduğunu göstermektedir.

Daha önce bahsedildiği üzere yapılan puan tahminleri için elde edilen sonuçlarda yakın olarak yapılmış olan puan tahminleri de başarılı tahmin olarak sayılmaktadır. Bu kapsamda Meta LLama dil modelinin verilen test veri setindeki yazılım hataları için yaptığı tahminlerin dağılımı şekil 16'da görüleceği şekilde olmuştur.



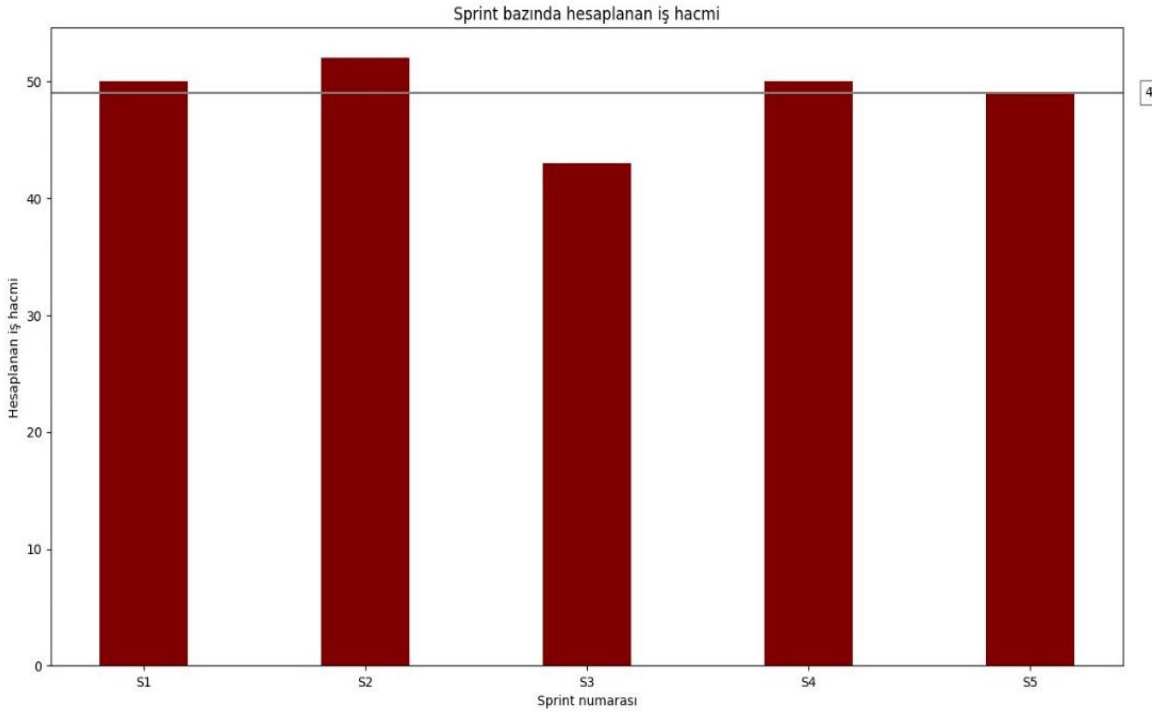
Şekil 19. Meta LLama puan tahminleri kabul kriteri dağılımı

Yine kabul kriterini göz önünde bulundurarak Meta LLama dil modelinin puan tahminindeki tutarlılığı değerlendirildiğinde, Şekil 17'de görülebileceği üzere %85,7 başarı sağlanırken %14,3 başarısız tahminde bulunulmuştur.



Şekil 20. Meta LLama kabul kriterli puan tahmin sonuçları

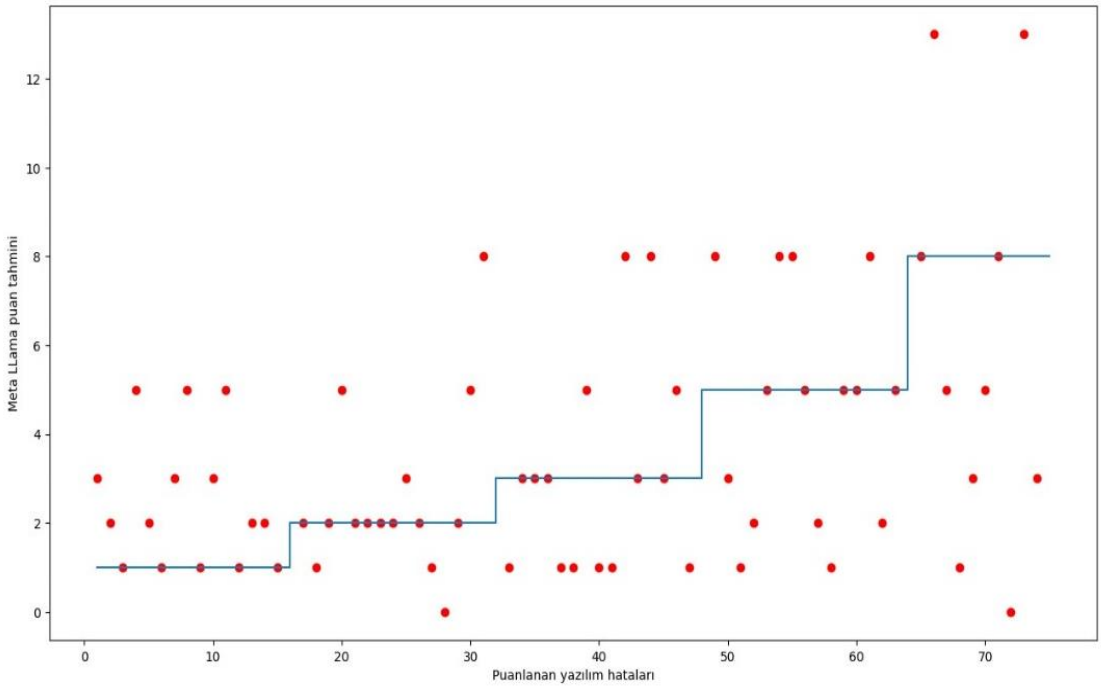
İkinci değerlendirme kriteri olan Velocity (iş hacmi) açısından Meta LLama'yı değerlendirecek olursak, hepsi 49 puanlık olacak şekilde oluşturulan 5 Sprint için yapılan testler sonucunda sırasıyla, 50 puan, 52 puan, 43 puan, 50 puan 49 puan olarak hesaplanmıştır. Bu sonuçlara göre 43 puan olarak hesaplanan Sprint haricinde şekil 18'de de görülebildiği üzere iş hacmi hesaplamasında da kabul kriterleri içinde kalarak %80 başarı sağlamıştır.



Şekil 21. Meta LLama sprint bazında hesaplanan iş hacmi

5.1.2 Phi 3

Meta LLama'nın ardından bire bir aynı testler Phi 3 dil modeli ile yapılmıştır. Önceki bölümlerde bahsedildiği üzere LM Studio üzerinde ayağa kaldırılan dil modeli önce eğitim veri seti ile eğitilmiş, ardından test edilmiştir. Test sırasında tek tek test veri setinin parçası olan yazılım hatalarının açıklamaları dil modeline gönderilmiş ve dil modelinden ilgili yazılım hatası için bir puan tahmininde bulunması beklenmiştir. Dil modelinden dönen tüm yanıtlar bir CSV dosyasına yazılmış ardından da sonuçların değerlendirilmesi aşamasına geçilmiştir. Phi 3 dil modelinin yaptığı puan tahminlerinin genel dağılımları Şekil 19'da görülebildiği gibi olmuştur.



Şekil 22. Phi 3 puan tahminleri dağılımı

Şekilden de anlaşılacağı üzere Phi 3 modeli de Meta LLama dil modeline benzer şekilde verilen test verilerinin bir kısmı için tam doğru puan tahmininde bulunurken bir kısmı tamamen gerçekten uzak tahminler olmuştur.

Tam puan tahmini, yakın puan tahmini ve tamamen yanlış puan tahmini şeklinde değerlendirdiğimizde Şekil 20'de gözlemlenebileceği üzere %35,7 tam doğru tahminde bulunurken, %24,2 oranında yakın tahminde bulunmuş, %40,1 oranında ise verilen test verileri için gerçekten uzak tahminlerde bulunmuştur.



Şekil 23. Phi 3 puan tahmin sonuçları

Tablo 2'de Phi 3 dil modeli için yapılan çalışma sonucunda elde edilen verilerin, sonuçları gözlemlenebilir.

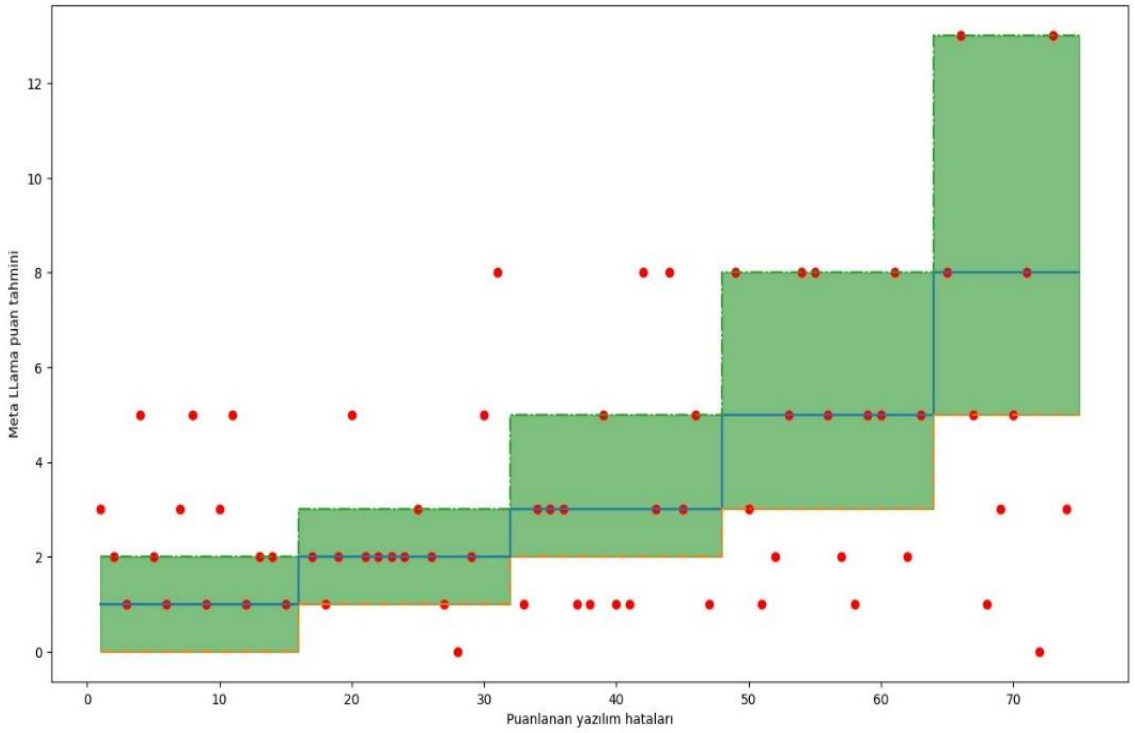
Tablo 2. Phi 3 dil modeli test sonuçları

	S1	S2	S3	S5	S8
Precisions	45,45	53,33	41,66	35,71	22,22
Recall	33,33	53,33	33,33	33,33	20
F1-Score	10,4	7,5	10,8	11,6	19

Bu sonuçlara göre Phi 3 dil modeli, genel olarak Meta LLama dil modeline nazaran düşük performans göstermiş olmasının yanında, Meta LLama dil modelinden farklı olarak yüksek puanlı yazılım hatalarında daha iyi performans göstermiştir. Veri setinin oluşturulması sırasında faydalanılan ve yazılım hatalarının elde edilmesinde kullanılan proje göz önüne alındığında, düşük puanlı yazılım hatalarının sayısının daha yüksek

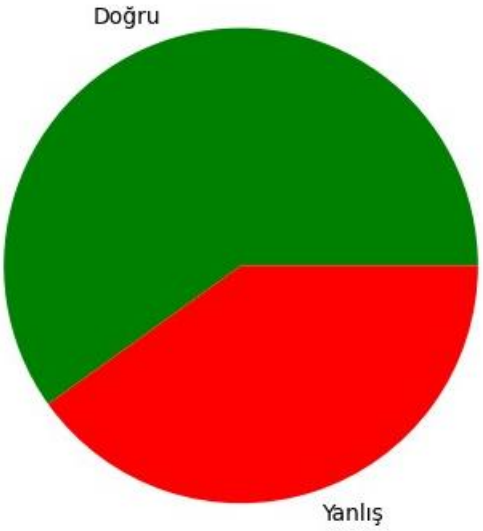
olmasından mütevellit bu sonuçların, daha az ihtiyaç duyulan alanda yüksek başarılı olduğunu göstermektedir.

Daha önce bahsedildiği üzere yapılan puan tahminleri için elde edilen sonuçlarda yakın olarak yapılmış olan puan tahminleri de başarılı tahmin olarak sayılmaktadır. Bu kapsamda Phi 3 dil modelinin verilen test veri setindeki yazılım hataları için yaptığı tahminlerinin kabul kriterlerine göre dağılımı Şekil 21'de gözlenebileceği şekilde olmuştur.



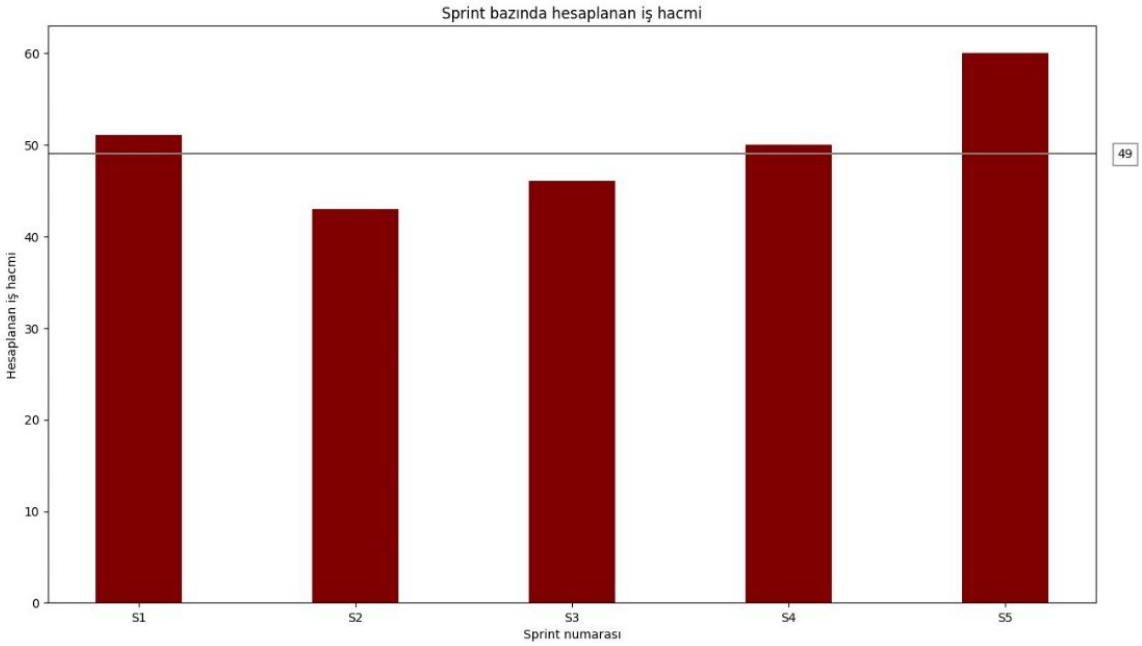
Şekil 24. Phi 3 puan tahminleri kabul kriteri dağılımı

Yine kabul kriterini göz önünde bulundurarak Phi 3 dil modelinin puan tahminindeki tutarlılığı değerlendirildiğinde, Şekil 22'de görülebileceği üzere %59,9 başarı sağlanırken %40,1 başarısız tahminde bulunulmuştur.



Şekil 25. Phi 3 kabul kriterli puan tahmin sonuçları

İkinci değerlendirme kriteri olan Velocity (iş hacmi) açısından Phi 3 dil modelini değerlendirecek olursak, hepsi 49 puanlık olacak şekilde oluşturulan 5 Sprint için yapılan testler sonucunda sırasıyla, 51 puan, 43 puan, 46 puan, 50 puan 60 puan olarak hesaplanmıştır. Bu sonuçlara göre Şekil 23'te de gözlemlenebileceği üzere, iş hacmi hesaplamasında da 3 sprintte kabul kriterleri içinde kalarak %60 başarı sağlamıştır.



Şekil 26. Phi 3 sprint bazında hesaplanan iş hacmi

5.2 Tartışma

Elde ettiğimiz bu sonuçlara göre Meta LLama dil modeli ile elde edilen sonuçlar Phi 3 dil modeli ile elde edilen sonuçlara göre çok daha başarılı olduğu görülmektedir. Literatürde benzer alanda yapılan çalışmalar incelendiğinde öncelikle yapılan çalışmanın bir metin sınıflandırma işlemi olduğuna dikkat çekmek gerekir. Seçilen veri setinde yazılım hatalarına ait tanımlamalar ve tekrarlama adımları metin halinde bulunmaktadır ve puan tahmini de aslında bu metinleri içerik karmaşıklıklarına göre sınıflandırma yapılması işlemidir.

Ülkemiz üniversitelerinde benzer alanda yapılan birçok çalışma bulunmaktadır. Bunlardan oran itibariyle daha başarılı sonuç elde edilenler incelendiğinde bunlardan birine örnek olarak Eyüp Halit Yılmaz'ın 2024 yılında yaptığı tez çalışması ele alınabilir. Bu çalışmada açık kaynak kodlu olarak geliştirilen Mozilla projesinin yazılım hatalarının önceliklendirilmelerine bu tezde yapılana benzer bir şekilde otomatize bir sistem ile ulaşılmaya çalışılmıştır. Yılmaz yaptığı çalışmada 5 farklı kategorideki yazılım hatalarından eşit sayıda seçerek bir veri seti oluşturmuştur. Çalışma sırasında BERT dil modelinden faydalanarak 5 öncelik grubunda sırasıyla %57,88, %42,72, %41,96, %45,24 ve %67,63 olmak üzere ortalamada %55,28 başarı sağlayabilmiştir (Yılmaz, 2024).

Yine benzer konuda büyük dil modelleri ile biyomedikal konulu metinlerde yararlı bilgilerin çıkartılması üzerine 2020 yılında çalışma yapmış olan Nur Bengisu Çam'ın çalışmaları da karşılaştırma için incelenebilir. Bu çalışmada da BERT tabanlı dil modelleri ile LLama dil modeli kullanılmıştır. Çalışma sonucunda kullanılan metinlerin kategorize edilmesinde kullanılan veri setlerinden birinde BERT tabanlı dil modellerinden BioBert ile %84,48, PudMedBERT ile %86,13, SciBERT ile %78,55 başarı sağlanırken, LLama %63,51 başarı sağlanabilmiştir (Çam, 2020).

Metin sınıflandırma çalışmalarının globalde de birçok örneği bulunmaktadır. Bunlardan biri olan Aleksandra Edwards ile Jose Camacho-Collados'un "Language Models for Text Classification: Is In-Context Leraning Enough?" çalışmasını inceleyecek olursak, bu çalışmada araştırmacılar dil modellerini sınıflandırma yapılmasının beklendiği konularda eğiterek metin sınıflandırma işlemi üzerinde bu metodun başarısını test etmek istemişlerdir. Bu çalışmada yine LLama dil modelinden de faydalanan araştırmacılar, en

başarılı alan olan IMDB film eleştirisi kısmında yapılmış olan yorumlar üzerinde yaptıkları çalışmada sınıflandırmada %62,8 başarı sağlamışlardır (Edwards ve Camacho-Collados, 2024).

Bu çalışmalar ile karşılaştırdığımızda Phi 3 dil modeli ile tez kapsamında yapılan çalışmada elde edilen %59,9 başarı ortalama bir seviyede iken, LLama dil modeli ile elde edilen %85,7 başarının oldukça iyi olduğu söylenebilir. Ayrıca metin sınıflandırma sonuçlarının diğer değerlendirme kriteri olan iş hacmi değerlendirmesinde de LLama dil modeli ile elde edilen başarı göz önünde bulundurulursa, projenin gerçek hayatta kullanım için de yeterli seviyede olduğu değerlendirilebilir.

6. SONUÇ ve ÖNERİLER

Tez çalışması kapsamında yazılım geliştirme metodolojilerinden biri olan Scrum'ın süreçlerinden biri olan planlama toplantılarının, yazılım geliştirme uzmanları üzerindeki zamansal yükünü hafifletmeye yönelik süreci otomatize edebilecek bir yöntem geliştirilmeye çalışılmıştır. Bu süreçte veri seti olarak Siemens Türkiye'de geliştirilmekte olan bir projeye ait gerçek yazılım hatalarından ve bu yazılım hataları için yazılım geliştirme uzmanları tarafından verilmiş olan puan tahminleri derlenmiştir.

Bu veri seti üzerinde yapılacak çalışma için lokal bir bilgisayarda LM Studio yardımı ile çalışmada kullanılan Meta LLama ve Phi 3 dil modelini kullanan serverlar ayağa kaldırılmıştır. Bu serverlara yine lokalde çalışmakta olan Python dilinde yazılmış kodlar ile gönderilen sorgu paketleri ile önce serverlarda çalışmakta olan büyük dil modellerinin eğitilmesi, ardından da eğitilmiş olan dil modellerinin test veri seti ile başarısının kontrol edilmesi sağlanmıştır. Eğitim sürecinin ardından gelen test sürecinde Phi 3 dil modeli ile yapılan testler daha ortalama bir sonuç verirken, LLama dil modeli ile yapılan denemeler sonucunda oldukça verimli sonuçlar alınmıştır.

Bu tez çalışması kapsamında dil modellerinin hedeflenen görevi yerine getirmekteki başarısı test edildiği için lokal bir bilgisayar üzerinde bu çalışma yapılmış olsa da, aynı dil modelleri benzer tekniklerle bir bulut ortamında ayağa kaldırılıp, internet ağı üzerinden yapılacak sorgular ile hem eğitim hem de eğitim sonrası gerçek görevde kullanımı sağlanabilir.

Tez çalışması sırasında kullanılan bilgisayarın CPU, RAM ve ekran kartı özelliklerinin ortalama düzeyde olması, eğitim sürecini çok uzatmış, eğitim sonrası test sürecinde ise soru-cevap şeklinde ilerlendiğinde kayda değer bir probleme sebep olmadığı gözlenmiştir. Dolayısıyla bu tez çalışmasının devamı olarak yapılacak çalışmalarda eğer proje bulut ortamına taşınacak olursa, kullanılan ortamın bu tarz donanım özelliklerine de dikkat etmek, çalışmanın başarısı üzerinde teorik olarak doğrudan bir etkisi olmasa dahi süre açısından verimliliği arttırabilir.

7. KAYNAKÇA

A. Lawal, R. C. (2021). *A Comperative Analysis of Agile and Waterfall Software Development Methodologies.* Bakolori Journal: 15.02.2023 tarihinde https://www.bakolorijournal.com/article/v11-2/assets/v11-2-5.pdf adresinden alındı

Angara, J. ve Prasad, S. (2020). *DevOps Project Management Tools for Sprint Planning, Estimation and Execution Maturity.* Sofia: Cybernetics and Information Technologies 20(2):79-92 http://dx.doi.org/10.2478/cait-2020-0018

Alistair Cockburn, D. C. (2016). *The Heart of Agile The Ordered Process for Improving Agile Engineering Outcomes: A Comparison of Commercial and Defense Software.* Cross Talk Online: 15.02.2023 tarihinde www.crosstalkonline.org/submission-guidelines adresinden alındı

Alpaydın, E. (2014). *Introduction to Machine Learning.* Cambridge: The MIT Press.

Baloğlu, E. B. (2023). *Reinforcement Learning for Text Classification: an Evalution of Policy-Gradient Methods with Various Topologies.* İstanbul: Sabancı University.

Bender, E. M., Gebru, T., McMillan-Major, A., ve Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency,* (s. 610-623).

Blum, A. (2007). *Machine Learning Theory.* ABD: Carnegie Mellon University.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ve Amodei, D. (2020). *Language models are few-shot learners. Advances in Neural Information Processing Systems.* Johns Hopkins University, OpenAI. https://doi.org/10.48550/arXiv.2005.14165

Cho, L. (2009). Adopting an Agile Culture A User Experience Team's Journey. *Agile Conference.* https://doi.org/10.1109/AGILE.2009.76

Çam, N. B. (2020). *Large Language Models for Biomedical Relation Extraction and Explanation.* İstanbul: Boğaziçi University.

D'Aveni, R. (1994). *Hypercompetition: Managing the Dynamics of Strategic Maneuvering.* New York: Free Press.

E. Cabello, J. C. (2017). *Communications in Computer and Information Science.* Berlin: Springer.

Edwards, A., ve Camacho-Collados, J. (2024). Language Models for Text Classification: Is In-Context Learning Enough? *Language Models for Text Classification: Is In-Context Learning Enough?* (s. 8-10). Torino: LREC-COLING. https://doi.org/10.48550/arXiv.2403.17661

Erdoğan, O, ve Pekkaya M. E. (2017). *More effective sprint retrospective with statistical analysis.* New York: Journal of Software: Evolution and Process. DOI:10.1002/smr.1933

Francesco Camastra, A. V. (2015). *Machine Learning for Audio, Image and Video Analysis.* Berlin: Springer.

Günerkan, M., Şahinaslan, E., ve Şahinaslan, Ö. (2022). Gümrük Beyannamesi Sürecinde Öğrenmeye Dayalı Algoritmaların Etkinliğinin İncelenmesi. *Acta Infologica, 6*(2), 175-188. doi:https://doi.org/10.26650/acin.1057060

Gürbüz, A. (2022). *Agile Scrum Methodology in Turkey and Implementation with Extreme Programming.* (Tez no: 724719). [Yüksek Lisans Tezi, Bahçeşehir Üniversitesi] https://tez.yok.gov.tr/UlusalTezMerkezi/tezSorguSonucYeni.jsp

Harrington, P. (2012). *Machine Learning in Action 1st Edition.* New York: Manning.

Hunt, J. (2006). *Agile Software Construction.* Berlin: Springer.

Ian Witten, E. F. (2006). *Data Mining: Practical Machine Learning Tools and Techniques.* ABD: Morgan Kaufmann.

Iqbal, M. (2022). *How the 12 principles in the Agile Manifesto work in real life.* Scrum.org: 15.02.2023 tarihinde https://www.scrum.org/resources/blog/how-12-principles-agile-manifesto-work-real-life adresinden alındı

Jeff Sutherland, K. S. (2020). *Scrum Guide.* Scrum Guides: 15.02.2023 tarihinde https://scrumguides.org/index.html adresinden alındı

Karla Mendes Calo, E. E. (2010). A Quantitative Framework for the Evaluation of Agile Methodologies. *Journal of Computer Science and Technology*.

Kartal, E. (2015). *Sınıflandırmaya Dayalı Makine Öğrenmesi Teknikleri ve Kardiyolojik Risk Değerlendirmesine İlişkin Bir Uygulama.* (Tez no: 394514). [Doktora Tezi, İstanbul Üniversitesi. İstanbul] https://tez.yok.gov.tr/UlusalTezMerkezi/tezDetay.jsp?id=Bezbp5vwVTSDwU473i2Y8A&no=wQEhnSu8MQKfgDPuNaiIMw

Kır, G. (2014). *Scrum Adoption in Kuvet Türk IT.* İstanbul: Doğuş Üniversitesi. 15.05.2023 tarihinde https://openaccess.dogus.edu.tr/xmlui/handle/11376/1047 adresinden alındı.

Leferve-Ardant, E. (2008, 10 7). *Is Scrum Evil?* Eric Lefevre-Ardant on Java & Agile:02.03.2024 tarihinde https://ericlefevre.net/wordpress/2008/10/07/scrum-is-evil/ adresinden alındı

Mance E. Harmon, S. S. (2000). *Reinforcement Learning: A Tutorial.* Centerville: Wright State University.

Matteo Golfarelli, S. R. (2012). Sprint Planning Optimization in Agile Data Warehouse Design. *Data Warehousing and Knowledge Discovery* (s. 30-41). Berlin: Springer, Berlin, Heidelberg.

McCormick, M. (2012). Waterfall vs Agile Methodology. *MPCS, inc*.

Mehryar Mohri, A. R. (2012). *Foundations of Machine Learning.* Cambridge: MIT Press.

Mike Beedle, A. v. (2001). *The Agile Manifesto.* Agile Manifesto: 10.02.2024 tarihinde www.agilemanifesto.org adresinden alındı

Mitchell, T. (1997). *Machine Learning.* New York: McGraw-Hill Education.

Nilsson, N. J. (2009). *The quest for artificial intelligence: a history of ideas and.* Cambridge; New York: Cambridge University Press .

P. Deemer, G. B. (2010). *The Scrum Premier.* 13.04.2024 tarihinde http://goodagile.com/scrumpremier/scrumpremier.pdf adresinden alındı

Rabia Saeed Malik, S. S. (2019). A Review of Agile Methodology in IT Projects. *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE) 2019.* 13.04.2024 tarihinde https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3351064adresinden alındı.

S. Goldman, R. N. (1995). *Agile Competitors and Virtual Organisations: Strategies for Enriching the Customer.* New York: Von Nostrand Reinhold.

Sahinaslan, E., ve Şahinaslan, Ö. (2019). Dijital dönüşümde öncelikli alanlar ve ilgili teknolojiler / Priority fields and related technologies in digital transformation. *International Congress on Business and Marketing.* s. 431-441.

Samar Alsaqqa, S. S.-N. (2020). Agile Software Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies*.

Sebastiani, F. (2002). *Machine learning in automated text categorization.* New York: ACM Computing Surveys.

Settles, B. (2009). *Active Learning Literature Survey* . University of Wisconsin Madison. 13.01.2024 tarihinde https://burrsettles.com/pub/settles.activelearning.pdf adresinden alındı.

Sheetal Sharma, D. S. (2012). Agile Processes and Methodologies: A Conceptual Study. *International Journal on Computer Science and Engineering*.

SU Jin-Shu, Z. B.-F. (2005). *Advances in Machine Learning Based Text Categorization.* Beijing: Institute of Software, Chinese Academy of Sciences.

Suryanarayana, G. (2015). Software Process versus Design Quality: Tug of Wa. *IEEE Software, 32*(4), 7-11. doi:DOI:10.1109/MS.2015.87

Şahin, M. (2016). *Agile (Scrum) Implementations for Banking Applications in Turkey, is Scrum usefull or not?* (Tez no: 436032). [Yüksek Lisans Tezi, İstanbul Bahçeşehir Universitesi İstanbul] https://tez.yok.gov.tr/UlusalTezMerkezi/tezDetay.jsp?id=8GBupnoh3fwAwBrrUw6XPQ&no=tbFlJYYsa_kKqpLzB1_glw

Şahinaslan, E. (2023). İş Süreci Optimizasyonu: Yöntem, Teknoloji, Riskler ve Fırsatlar. *Akademik İzdüşüm Dergisi, 8*(2), s. 570-604.10.01.2024 tarihinde https://dergipark.org.tr/tr/pub/beuiibfaid adresinden alındı

Şahinaslan, E., Günerkan, M., ve Şahinaslan, Ö. (2023). Makine Öğrenmesinde Kategorik Veri Kodlama Tekniğinin Kullanımına Alternatif Bir Çözüm Yöntemi. *Journal of Intelligent Systems: Theory and Applications, 6*(1), Şahinaslan, E. , Günerkan, M. ve Şahinaslan, Ö. (2023). Makine Öğrenmesinde Kategorik Veri Kodlama Te1-11. doi:DOI: 10.38016/jista.1140499

Şahinaslan, Ö., Dalyan, H., ve Şahinaslan , E. (2022). Naive Bayes Sınıflandırıcısı Kullanılarak YouTube Verileri Üzerinden Çok Dilli Duygu Analizi. *15*(2), s. 221-229. doi: https://doi.org/10.17671/gazibtd.999960

Şahinaslan, Ö., Şahinaslan, E., ve Küçükali, E. (2023). Üçüncü taraf bileşenlerden kaynaklı zayıflıkların tespiti ve yönetimine ilişkin bir uygulama. *Denetişim*(28), s. 53-74. doi:https://doi.org/10.58348/denetisim.1225325

T. C. Krehbiel, P. A. (2017). The Journal of Effective Teaching an online journal devoted to teaching excellence. *The Journal of Effective Teaching*, Vol. 17, Number 2.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., vd. (2017). *Attention is all you need. Advances in Neural Information Processing Systems.*

Long Beach, CA, USA.: 31st Conference on Neural Information Processing Systems (NIPS 2017).

Yılmaz, H. Y. (2024). *Automated Priority Detection in Software Bugs: A Comprehensive Study on Transformer-Based Encoders with Contrastive Learning, Large Language Models and Vector Databases for Enhanced Efficiency.* Ankara: Middle East Technical University.03.01.2024 tarihinde https://open.metu.edu.tr/handle/11511/108485 adresinden alındı.

Zakrisson, J. (2022). *Increasing Agile team efficiency by improving Sprint estimation accuracy.* Umeå: Umeå University, Faculty of Science and Technology, Department of Applied Physics and Electronics. 05.02.2024 tarihinde https://www.diva-portal.org/smash/get/diva2:1696480/FULLTEXT01.pdf adresinden alındı.

ÖZ GEÇMİŞ

Yekta Buğrahan NİZAMOĞLU

Eğitim

Yüksek Lisans	2024	Maltepe Üniversitesi, Bilgisayar Mühendisliği
Lisans	2019	İstanbul Teknik Üniversitesi, Kontrol ve Otomasyon Müh

Kariyer

Yıl	Firma	Görev
2019-Halen	Siemens Türkiye	Yazılım Geliştirme Uzmanı – Scrum Master