# Yazılım Güvenliğinde Derin Öğrenme Tabanlı Kaynak Kod Analizi ve Yorum Önerimi

Yusuf Kartal

**DOKTORA TEZİ**

Bilgisayar Mühendisliği Anabilim Dalı

Haziran 2023

# Deep Learning Based Source Code Analysis and Review Recommendation in Software Security

Yusuf Kartal

**DOCTORAL DISSERTATION**

Department of Computer Engineering

June 2023

# Yazılım Güvenliğinde Derin Öğrenme Tabanlı Kaynak Kod Analizi ve Yorum Önerimi

Yusuf Kartal

Eskişehir Osmangazi Üniversitesi
Fen Bilimleri Enstitüsü
Lisansüstü Eğitim ve Öğretim Yönetmeliği Uyarınca
Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Yazılımı Bilim Dalında
DOKTORA TEZİ
olarak hazırlanmıştır

Danışman: Prof. Dr. Kemal Özkan

Haziran 2023

# ONAY

Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Yazılımı Bilim Dalı Doktora öğrencisi **Yusuf Kartal**'ın DOKTORA tezi olarak hazırladığı "**Yazılım Güvenliğinde Derin Öğrenme Tabanlı Kaynak Kod Analizi ve Yorum Önerimi**" başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğin ilgili maddeleri uyarınca değerlendirilerek oybirliği ile kabul edilmiştir.

**Danışman** : Prof. Dr. Kemal Özkan

**İkinci Danışman** :

**Doktora Tez Savunma Jürisi:**

**Üye :** Prof. Dr. Kemal Özkan

**Üye :** Doç. Dr. Nihat Adar

**Üye :** Doç. Dr. Ahmet Arslan

**Üye :** Prof. Dr. Eyyüp Gülbandılar

**Üye :** Dr. Öğr. Üyesi Burcu Yılmazel

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Prof.Dr. Hürriyet ERŞAHAN
Enstitü Müdürü

# ETİK BEYAN

Eskişehir Osmangazi Üniversitesi Fen Bilimleri Enstitüsü tez yazım kılavuzuna göre, Prof. Dr. Kemal Özkan danışmanlığında hazırlamış olduğum "**Yazılım Güvenliğinde Derin Öğrenme Tabanlı Kaynak Kod Analizi ve Yorum Önerimi**" başlıklı Doktora tezimin özgün bir çalışma olduğunu; tez çalışmamın tüm aşamalarında bilimsel etik ilke ve kurallara uygun davrandığımı; tezimde verdiğim bilgileri, verileri akademik ve bilimsel etik ilke ve kurallara uygun olarak elde ettiğimi; tez çalışmamda yararlandığım eserlerin tümüne atıf yaptığımı ve kaynak gösterdiğimi ve bilgi, belge ve sonuçları bilimsel etik ilke ve kurallara göre sunduğumu beyan ederim.
23/06/2023

Yusuf Kartal

# ÖZET

Modern kod incelemesi, güvenliği sağladığı, hataları erken tespit ettiği ve kod kalitesini iyileştirdiği için yazılım geliştirme süreçlerinde kritik bir adımdır. Ancak, manuel incelemeler zaman alıcı ve güvenilmez olabilmektedir. Otomatik kod incelemesi bu sorunları çözebilir. Kod inceleme yorumlarını önermek için önerilmiş derin öğrenme yöntemleri olsa da, bunların eğitilmesi ve çalıştırılması maliyetlidir. Bunun yerine, otomatik kod incelemesi için bilgi erişim tabanlı yöntemler verimlilik, etkililik ve esneklik açısından umut verici sonuçlar sergilemektedir. Ana hedef, otomatik kod incelemede en iyi sonuçları veren vektöre dönüştürme yöntemi ile benzerlik yönteminin optimal kombinasyonunu belirlemek ve böylece bilgi erişim tabanlı yöntemlerin performansını ölçmektir. Ayrıca ön işlemlerin modellerin başarısı üzerindeki etkisini incelemek de hedefler arasında bulunmaktadır. Önceki araştırmalardan (TF-IDF ve Bag-of-Words) farklı olan birden fazla vektörleştirme yöntemi (Word2Vec, Doc2Vec ve Transformer) ve benzerlik yöntemi (Kosinüs, Öklid ve Manhattan) kaynak kod metinleri arasındaki anlamsal benzerlikleri belirlemek için çalışmaya dahil edilmiştir. BLUE, METEOR ve ROUGE-L gibi standart metrikleri kullanarak bu yöntemlerin performansı değerlendirilmiş ve modellerin çalışma süreleri de sonuçlara dahil edilmiştir. Elde edilen sonuçlara göre Transformer modeli tüm standart metriklerde ve benzerlik ölçümlerinde son çalışmalara göre daha iyi performans göstermektedir. Ayrıca tam eşleşme sağlamada %19,1'lik ve benzer öneriler sağlamada %6,2'lik bir iyileşme görülmektedir. Elde edilen bulgular, transformer modelinin, insanlar tarafından yazılanlara çok benzeyen kod inceleme yorumları önermek için oldukça etkili ve verimli bir yaklaşım olduğunu, otomatik kod inceleme sistemleri geliştirmek için değerli bilgiler sağladığını göstermektedir.

**Anahtar Kelimeler:** Modern Kod Gözden Geçirme, Kod İnceleme, Kod Benzerliği, Bilgi Erişimi, Vektörleştirme, Vektör Farkı

# SUMMARY

Modern code review is a critical step in software development as it ensures security, detects errors early and improves code quality. However, manual reviews can be time-consuming and unreliable. Automatic code review can fix these issues. While deep learning-based technics are proposed for recommending code review comments, they are costly to train and run. Instead, information retrieval-based methods for automated code review show promising results in efficiency, effectiveness, and flexibility. The main objective is to determine the optimal combination of the vector conversion and similarity methods that gives the best results in automatic code review, thus measuring the performance of information retrieval-based methods. It is also among the objectives to examine the effect of preprocessing on the success of the models. Different from previous studies (TF-IDF and Bag-of-Words), multiple vectorization methods (Word2Vec, Doc2Vec, and Transformer) and similarity methods (Cosine, Euclidean, and Manhattan) were included in the study to determine semantic similarities between source codes. The performance of these methods was evaluated using standard metrics such as BLUE, METEOR, and ROUGE-L, and the running times of the models were also included in the results. According to the results, the Transformer model performs better in all standard metrics and similarity measurements than in recent studies. In addition, there is an improvement of 19.1% in providing an exact match and an improvement of 6.2% in providing similar recommendations. The findings show that the transformer model is highly effective and efficient in suggesting code review comments similar to those written by humans, providing valuable information for developing automated code review systems.

**Keywords:** Modern Code Review, Code Review, Code Similarity, Information Retrieval, Vectorization, Vector Distance

# TEŞEKKÜR

Bu tez çalışması başta olmak üzere içinde bulunduğumuz tüm çalışmalarda benden desteğini esirgemeyen, beni yönlendiren ve her daim bir çıkış yolu bulmamda yardımcı olan çok değerli hocam ve danışmanım Sayın Prof. Dr. Kemal Özkan'a sonsuz saygı ve teşekkürlerimi sunarım.

Doktora tez izleme toplantılarında bilgi birikimleriyle tezime katkıda bulunan ve en iyi şekilde yönlendiren tez komitesi hocalarım Sayın Doç. Dr. Nihat Adar'a ve Sayın Doç. Dr. Ahmet Arslan'a teşekkürlerimi sunarım. Eğitim-öğretim hayatım boyunca üzerimde emeği olan burada adını sayamadığım tüm değerli hocalarıma minnetarım.

Doktora eğitimim süresince akademik ve teknik konuların yanı sıra maddi, manevi tüm konularda yanımda olan ve beni destekleyen Bilgisayar Mühendisi Dr. Savaş Okyay ve Bilgisayar Yüksek Mühendisi Merve Ceyhan ile birlikte başta Bilgisayar Yüksek Mühendisi Furkan Avcı ve Bilgisayar Yüksek Mühendisi Eyüp Kaan Akdeniz olmak üzere tüm Avkar Yazılım çalışanlarına teşekkür ederim.

Gösterdiği anlayış, sevgi ve destek ile beni motive ederek hayatıma mutluluk katan sevgili eşim Tuğba Kartal'a; varlığıyla ve sarılmalarıyla bana güç veren sevgi dolu oğlum Batu Kartal'a sonsuz teşekkür ederim.

Tez süresince bana ve aileme verdikleri destek ile güç toplamamızı sağlayan eşimin anne ve babasına; tüm yaşamım boyunca desteğini esirgemeyen ve bugünlere gelmeme vesile olan kardeşlerime, anneme ve babama minnettarım.

*Bu tezi sevgili anneme ve babama ithaf ediyorum...*

Yusuf Kartal

# İÇİNDEKİLER

# ŞEKİLLER DİZİNİ

# ÇİZELGELER DİZİNİ

**Çizelge** **Sayfa**

# SİMGELER VE KISALTMALAR DİZİNİ

| Simgeler | Açıklama |
|---|---|
| $\sigma$ | Standart Sapma |
| $\Delta$ | Fark |
| $\Delta$d | Uzaklık Farkı |

| Kısaltmalar | Açıklama |
|---|---|
| ACC | Doğruluk |
| AST | Abstract Syntax Tree |
| BERT | Bidirectional Encoder Representations from Transformers |
| BoW | Bag of Words |
| CFG | Control Flow Graph |
| CNN | Convolutional Neural Network |
| FP | False Positive |
| FN | False Negative |
| GNN | Graph Neural Network |
| CWE | Common Weakness Enumaration |
| DDoS | Distributed Denial of Service |
| KGG | Kod Gözden Geçirme |
| KL | Kullback-Leibler |
| k-NN | k Nearest Neighbor |
| LSTM | Long Short-Term Memory |
| NLP | Natural Language Processing |
| PPV | Kesinlik |
| RNN | Recurrent Neural Network |
| SQL | Structured Query Language |
| TP | True Positive |
| TN | True Negative |
| TPR | Duyarlılık |
| Tree-LSTM | Tree Long Short-Term Memory |
| XSS | Cross Site Scripting |

# 1. GİRİŞ VE AMAÇ

Yazılım geliştirirken özellikle henüz tecrübe kazanmamış yazılım geliştiriciler tarafından ana senaryo üzerinde çalışan bir yazılımın hatasız olduğu düşünülebilir ancak ileride gelecek değişiklikler diğer kod parçalarını da olumsuz etkileyerek bozabilmektedir. Bu şekilde kullanıcı karşısına çıkan üründe kullanıcılar ana senaryo dışına olağan bir şekilde çıktıklarında ortaya çıkan hatalar müşteri memnuniyetini düşürürken yazılım maliyetlerini oldukça arttırmaktadır. Hatanın kaynağının tespiti ve bu değişikliğin hatasız bir şekilde yapılması zor ve zaman alıcı bir süreçtir. Bu sebeple hataların canlı ortama çıkmadan önce tespit edilmesi için yazılım ekipleri çeşitli yöntemler geliştirmeye çalışmaktadırlar. Bu amaçla yazılım mühendisliğinin önceliği güvenli bir yazılım üretirken yazılım kalitesini yükseltmek, yazılım maliyetini düşürmek, yazılımın bakım ve gelişimini kolaylaştırmaktır. Yazılım mühendisleri bunları hedefleyerek yazılımın karmaşıklığını azaltıp anlaşılabilirliğini arttırmak, gelişimini ve bakımını kolaylaştırmak için uygulanabilecek yazılım mühendisliği yöntemleri arayışı içindedir. Yazılım mühendisliğinde kod kalitesinin yükseltilmesine yönelik çeşitli yöntemler geliştirmek için yapılan çalışmaların tarihi modern programlama dilleri ile neredeyse aynı tarihe dayanmaktadır. Sıklıkla kullanılan ve yazılımdaki uygunsuzlukların düzeltilmesinde en etkili yollardan birisi olarak nitelendirilebilecek yöntemlerden biri olan kod incelemesi, meslektaş incelemesi diye de adlandırılan kod gözden geçirme (KGG) yazılım geliştirme sürecinin önemli bir parçasıdır. Bir veya bir kaç kişinin temel olarak kaynak kodun bölümlerini görüntüleyerek ve okuyarak bir yazılımı kontrol ettikleri düzeltme önerdikleri veya bu kod parçasını reddettikleri bir kalite güvence aktivitesidir. KGG'i gerçekleştiren kişilerden en az birinin ilgili kod parçasının geliştiricisi olmaması gerekir (Baum vd., 2016a; Huizinga ve Kolawa, 2007). KGG, her ne kadar doğrudan kod kalitesi ile ilgili gibi olsa da genellikle farklı amaçlar için de yapılmaktadır (Bacchelli ve Bird, 2013; Baum vd., 2016b). Örneğin, KGG ile hatalı kod parçalarının yanı sıra kötü amaçlı yazılım davranışları, bellek taşmaları, aşırı bellek kullanımları, senkronize çalışan kodlar arasında kilitlenmeler ve biçimsel metin bozuklukları da tespit edilebilmektedir böylece KGG, geliştirilen yazılımın sürdürülebilirliğini sağlamak amacıyla yapılabilirken doğruluk, performans, güvenlik ve güvenilirlik gibi kusurların tespit edilmesinde de önemli bir rol oynamaktadır. KGG ile deneyimli yazılımcıların bilgi ve tecrübeleri de deneyimsiz yazılımcılara aktarılabilmektedir böylece yazılımcılar arasında yazılan kodu

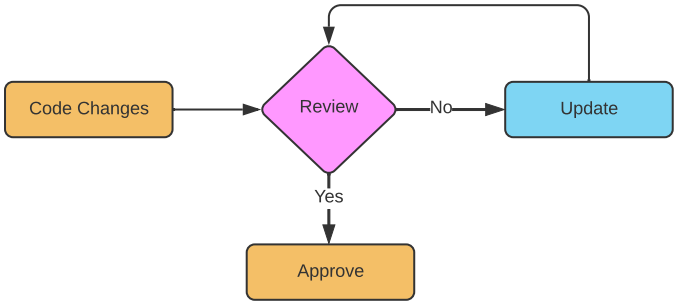
sahiplenme, karşılıklı sorumluluk duygusu ve dayanışma artmaktadır. Bunlara ilaveten yeni ve daha iyi çözümler üretilebildiği gibi üzerinde birlikte çalışılan yazılımda yeni fikirlerin ortaya çıkması gibi faydaları da bulunmaktadır. Hava trafik yazılımlarında veya kurum kültürü olarak tüm yazılımlarında gözden geçirmeyi zorlayan şirketlerde olduğu gibi bazı özel alanlarda kalite güvence kurallarını sağlayabilmek için KGG zorunluluk olarak karşımıza çıkabilmektedir. KGG, statik ve dinamik olmak üzere 2 farklı yaklaşımla yapılabilmektedir. Statik KGG, kodu çalıştırmadan ve çoğunlukla yapısal olarak inceleyerek en iyi yaklaşımların uygulanmaya çalışıldığı yöntemdir. Dinamik KGG ise yazılımın çalışma mantığının da incelenmek üzere kimi zaman doğrudan çalıştırılarak canlı olarak mercek altına alındığı yöntemdir. Statik KGG ile bir kod satırındaki belli bir kural dikkate alınırken dinamik KGG kodun mantığına odaklanmaktadır. Statik KGG için tanımlanabilir kurallar söz konusu olduğundan günümüzde bu konuda bilinirliği yüksek ve yaygın olarak kullanılan (örn. SonarQube) uygulamalar bulunmaktadır. Dinamik KGG, üzerinde çalışılan kodun ne yaptığının anlaşılmasını gerektirdiğinden daha zahmetli ve daha dikkatli bir süreç içermektedir. Dinamik KGG için planlanan kod gözden geçirme toplantılarında geçmiş tecrübeler ile tespit edilebilecek kodlama stilleri ya da en iyi yaklaşımların ihlali gibi uygunsuz kod parçalarının varlığı dikkatin dağılmasına ve gözden geçirme kalitesinin düşmesine sebep olabilmektedir (Anshul Gupta ve Sundaresan, 2018). Bununla birlikte tecrübeli yazılımcıların aynı veya benzer tür hatalar ile sürekli karşılaşmaları sıkıcı bir durum oluşturmakla birlikte vakit kayıplarına sebep olmaktadır. Bu kapsamda Baca vd. tarafından yapılan çalışmada statik analiz araçlarının kullanılması durumunda yazılım projesi maliyetinden %17 oranında tasarruf sağlanabileceği belirtilmiştir (Baca vd., 2008). Jones tarafından 2015 yılında yapılan detaylı çalışmada 1394 uygunsuz kaynak kod barındıran bir yazılım projesi üzerinde uygulanan analiz, geliştirme ve test süreçlerinde kod gözden geçirme aşamasında 1071 hata tespit edilmiştir. Ardından yapılan statik analiz sürecinde geriye kalan 157 uygunsuzluktan 136 tanesi tespit edilmiştir. Bu şekilde çalışma ortamına sunulan yazılımlarda ilk 30 günde karşılaşılabilecek hata oranının %99'un üzerinde bir oranla düştüğü görülmüştür. Bu durumun yazılımın geliştirme ve bakım aşamalarında ciddi maliyet tasarrufları sağladığı bildirilmiştir (Jones, 2017).

Yaşanan teknolojik değişimlere bağlı olarak yazılım geliştiricilerin yazılım geliştirme süreçlerinden beklentileri de değişmiştir. Geliştirilen yazılımların kalitesi, beklenen performansı ve gerçekleştirilebilecek yaklaşımlar üzerine yapılan çalışmalar hızla popüler hale gelmiştir. Kaydedilen ilerleme ile büyük ölçekli projelerde zaman yönetimi bir öncelik halini almıştır (Barki vd., 1993).

Geliştiricilerin karşılaştıkları kodlama hatalarını kısa sürede çözmek, zaman yönetimi sağlamada esastır. Geliştirme aşamasında geliştiriciler tarafından yapılan basit hatalar önemsiz görünse de, kod satırları arttığında önemli güvenlik sorunlarına dahi yol açabilmektedir (Chess ve McGraw, 2004). Daha önceleri yazılım geliştirme süreçlerine odaklanılırken artık hataların daha hızlı fark edildiği yaklaşımlar ve kodlarda uyumluluk, birleştirme, okunabilirlik, karmaşıklık gibi kalite özellikleri önem kazanmıştır (Pantiuchina vd., 2018). Bu nedenle projelerde etkin zaman yönetiminin sağlanabilmesi için geliştiricilerin karşılaştığı hatalara hızlı ve doğru bir şekilde çözüm bulunması gerekmektedir. Geliştirilen yazılım içerisinde test aşamasında veya kullanıcıya teslim edildikten sonra yani erken aşamada fark edilemeyen hatalar ortaya çıkabilir. Bu durum geliştiricilere zaman ve maddi kayıp olarak yansımaktadır. Bu sebeple KGG, bir uygulamada ortaya çıkabilecek hataların ya da uygunsuzlukların sahaya yansımadan önce tespit edilebileceği en önemli aktivitelerden biridir. Bunun yanı sıra yazılım standartlarının belirlenmesi ve bu standartlara uyulması, yardımcı araçlar ile denetlenen kod yazma rutinleri, geliştirme sürecinde yazılan testler, kod incelemeleri ve geliştirme tamamlandıktan sonra otomatik kontrol araçlarıyla uygulamanın denetlenmesi gibi bir çok yöntem kod kalitesinin arttrılması için uygulanmaktadır. Bu aktiviteler kod kalitesini artırmak için bilinen birincil işlemlerdir (H. W. Kim, 2020). Bu sürecin en fazla insan emeği gerektiren kısmı yazılım geliştirme ekibi tarafından gerçekleştirilen KGG'dir.

**Modern Kod Gözden Geçirme**: KGG, potansiyel hataları erkenden tespit etmek, potansiyel iyileştirmeler için daha doğru altyapılar oluşturmak, kodlama yaklaşımlarını iyileştirmek ve yazılım geliştiricileri arasında bilgi yaymak için gerekli hale gelmiştir (Cunha vd., 2021). Bu sebeple KGG yazılım mühendisliğinde son yıllarda önerilen bir uygulama olarak kabul edilmektedir. Hem ticari olarak hem de açık kaynak kod sistemlerinde yazılım hatalarının erken tespiti ve azaltılması için kullanılmakta olan yaygın bir süreçtir. Konfigürasyon yönetim araçlarının daha verimli kullanılmasıyla birlikte, önceden sadece seçilen bir kod parçası üzerinde toplantılar şeklinde gerçekleşen kod inceleme süreçleri, diğer geliştiricilerin eklenen tüm kod satırlarını gözden geçirebildiği, geri bildirimlerin iletilebildiği ve ekip onayıyla sisteme dahil edildiği bir sürece dönüşmüştür. Bu haliyle modern KGG, geliştirme aşamasında çıkan sorunları tespit etmeyi ve ortadan kaldırmayı amaçlayan basit, eşzamansız ve araç destekli bir tekniktir (Bacchelli ve Bird, 2013). Uzunca bir süredir çeşitli ortamlarda açık kaynak kodlar yazılım geliştiriciler tarafından yayınlanmakta ve ilgilenen diğer yazılım geliştiriciler tarafından gözden geçirilerek düzenlemeler önerilmektedir. Bu hızlı geri beslemeler içeren paralel süreç ile kod kalitesi çarpıcı derecede iyileşmiştir.

Hem endüstriyel hem de açık kaynak projelerde yazılım sistemlerinin kalitesini artırmanın bir yolu olarak modern KGG günlük bir rutin olarak benimsenmiştir (Uchoa vd., 2020). Bu şekilde emek yoğun bir hale gelen modern KGG için uygulanması gereken adımlar satır satır yapılan kod incelemelerine dayandırılarak oldukça sıkı bir çerçevede formülleştirilmiştir. Bu formül etkili, uygulanabilir ancak sistemi yavaşlatan yapıya sahip bir süreçtir.




Şekil 1.1. Modern kod gözden geçirme akışı

Şekil 1.1.'de resmedilmiş Kaynak Kod Yaşam Döngüsü içerisinde yer alan Çekme İsteği (Pull Request) adımları şu şekilde açıklanabilir; (Thongtanunam vd., 2015)

1. Bir yazılım geliştirici kaynak kod üzerinde bir değişiklik yapar ve gözden geçirilmesi için sisteme gönderir.
2. Diğer yazılım geliştiricileri ya da ilgili kişileri, eklenen ya da değiştirilen kod parçasını gözden geçirip gerekiyorsa yorumlamaları için davet eder.
3. Davet edilen kişiler ekip içindeki görevine göre değişikliği inceleyerek çeşitli düzeltmeler önerebilir ya da bazı kısımları tartışmaya açabilir. Aynı zamanda gerçekleştirilen yazılım geliştirme görevinin yerine getirilip getirilmediği ve sistemde başka problemlere yol açmadığını garantilemek için ilgili testleri yürütebilir. Bu süreç sonucunda uygunsuzluklar işaretlenerek ve sebepleri belirtilerek yazılım geliştiricinin düzeltmesi talep edilir. Yazılım geliştirici bu geri beslemeler ışığında yeniden düzenlediği kodları tekrar sisteme gönderir. Bu adım, davet edilen kişiler bu kaynak kod değişikliklerini uygun bulup onaylayıncaya kadar devam eder.

4. Birden fazla kişi tarafından incelenen ve uygunluk onayı alan kaynak kod değişikliği ana kod deposuna birleştirilir, eğer uygunluk onayı alınamadıysa reddedilerek ana kod deposunda hiç bir değişiklik yapmaması sağlanır.

Böylece Modern KGG ile gözden geçirilmemiş kodların ana dizine uygulanması engellenerek hataların erken tespit edilmesi hedeflenir. Tüm değişikliklerin ardından uygun olarak onaylanan kaynak kod değişikliklerinin kod deposu ana dizinine aktarılması sağlanır. Bu şekilde yazılım geliştirme süreçlerine katkı sağlayarak kod kalitesini arttıran ve son kullanıcıya ulaşan üründeki hata oranını azaltan bir süreç ortaya konmuş olunur.

Yazılım geliştirme sürecinin etkili, gerekli, emek yoğun ve vazgeçilmez bir adımı haline gelen Modern KGG'nin yan etkisi olarak karşımıza çıkan geliştirme faaliyetlerinin yavaşlamasına sebep olması ile son yıllarda topluluklar ve açık kaynak kod sistemleri bu süreci hızlandırmak için daha az gereklilikler ile kod gözden geçirme sürecini uygularken çeşitli araçlardan da destek almaya başlamışlardır. Uzman bir kişi gibi mevcut kod depolarından öğrenip kaynak kodlar üzerinde düzeltme önerilerinde bulunan bir sistem daha fazla bulgunun tespit edilmesine imkan verebilecektir. Yıllardır, yazılımlarda otomatik olarak hata bulma ile ilgili araştırmalar yapılmaktadır (Bessey vd., 2010). Derleyiciler sadece kodların yapısal olarak doğruluğunu kontrol etmektedirler. Statik kod analizi ile belirlenen programlama kuralları kaynak kod üzerinde eşleştirilmekte ve birçok kural ihlali tespit edilebilmektedir (Bessey vd., 2010). Geleneksel diyebileceğimiz statik analiz yöntemi kullanıcılar ya da sağlayıcılar tarafından tanımlanması gereken kurallar gerektirmektedir. Gerçek bir yazılım projesi için kullanıcıdan beklenen özelleştirmelerin yapılması zor bir görev içermekte ve çoğu zaman yapılmamaktadır. Bu zor süreç için sağlanabilecek en iyi çözüm otomatik olarak öğrenen bir sistem olabilir. Geniş ölçekli açık kaynak kod geliştirme yapılarının ve depolarının varlığı bu fikri mümkün kılmaktadır. Bu yapıların analiz edilmesi ile gerçek yazılım projesi örneklerinin hangi bileşeni ne şekilde kullandığı ile ilgili sonuçlar ortaya konabilir (Murali vd., 2017). Otomatik olarak yazılım içindeki uygunsuzlukları tespit eden bir uygulama yazılımcının bu hataları düzelterek gözden geçirme işlemini yapan kişilerin daha detaylı hatalara odaklanmalarına destek olacaktır. Bazı statik kod analiz araçları, otomatik kod incelemesine yardımcı olmak için kullanılmaktadırlar. Bu statik analiz araçları daha hızlı ve verimli bir şekilde bulgular üretirken gözden geçirme sonucunda elde edilen çıktılar ile karşılaştırmak doğru bir yaklaşım olmayacaktır. Zira sürekli öğrenen bir sistem belirli kurallar ile durağan bir şekilde sürekli aynı çıktıları vermekten öte

değişen yaklaşımları ve güvenlik açıklarını saptayıp çözümler sunma noktasında büyük fayda sağlayacaktır. Öte yandan yazılım geliştirme aşamasında kodlama yapan geliştirici ekip üyeleri bazı temel prensipleri benimseyerek kodlardaki güvenlik açıklarını kapatmakta ve bu pratikleri tüm yazılım geliştirme ekibine yaymak istemektedirler. Bu pratikler ekibe yeni katılan bir üye için hem global çapta hem de kurumsal çapta aktarılması kolay olmayan parametreler içermektedir. Bu parametreler çoğunlukla tecrübeli yazılım geliştiricilere danışılarak ya da danışılmadığı durumlarda eğer yapılıyorsa kod gözden geçirme toplantılarında ya da tecrübeli geliştiricilerin yorumları ile değilse hatalar ya da güvenlik ihlalleri ortaya çıkınca öğrenilmektedir. Hem eğitim aşamasında hem de uygulama aşamasında yazılım geliştiriciye bu pratikleri kazanmasında destek olacak tecrübeli bir yol gösterici şüphesiz kod kalitesini arttırırken oluşabilecek yazılım hatalarının ve güvenlik açıklarının da önüne geçecek ve tecrübeli yazılım geliştiricilerin vaktinden çalmadan yeni geliştiriciler için büyük farkındalık ve kolaylıklar sağlayacaktır.

Bu bilgiler çerçevesinde geçmiş kod gözden geçirme aktivitelerinden beslenerek öneriler sunacak bir sistemin amaçlanan kullanım alanları ve durumları şu şekilde sıralanabilir;

1. Yazılım şirketleri açısından işe yeni başlayan yazılımcıların şirket içi yazılım eğitimlerinin daha hızlı ve takip edilebilir olmasını sağlayabilecektir. Ekibin benimsediği yazılım geliştirme standartlarına uyulmaması durumunda işe yeni başlayanı uyaran bir sistem olarak ekibin ayıracağı zamandan tasarruf sağlayacaktır.
2. Özellikle yeni saptanan ve henüz bir çok araç tarafından saptanmayan ama tecrübeli yazılım geliştiriciler tarafından bilinen güvenlik açıklarının canlı ortama çıkmadan kapatılmasını sağlayabilir.
3. Programlama eğitimi süresince öğrencilerden çeşitli görevler için kod yazmaları istenmektedir. Birçok yazılım geliştirme ortamı temel olarak kod yazma sürecine destek olsa da eğitmen rolüne benzer olarak öneride bulunan bir eklenti ders sırasında dersi veren eğitmenin yükünü azaltarak daha çok bilgi aktarımı sağlayabilir.
4. Kod kalitesinin arttırılması için eğitilmiş sanal bir geliştirici ya da daha doğrusu uzman kod gözden geçiricisi ile kod yazan yazılım geliştiriciler tarafından daha kaliteli, güvenli ve hatasız kodlar üretilmesi hedeflenmektedir.

Bu tez kapsamında deneyimli yazılım geliştiriciler tarafından incelenmiş ve çeşitli sorunlar tespit edilerek değişiklik isteğinde bulunulmuş ve farklı geliştiriciler tarafından yorumlar yapılmış kod parçaları üzerinde çalışılmıştır. Eğitilen bir yapay öğrenme modelinin bu incelemelerden en uygun olanı ya da olanları önerebilmesini sağlamak için deneyler yapılmıştır. Bu süreçte veri temizleme, metin sayısallaştırma gibi çeşitli aşamalarda farklı yöntemlerin uygulanması ve en başarılı sonuç veren kombinasyonların tespit edilmesi sağlanmıştır. Doğal dil işleme konusunda uygulanan farklı yöntemlerin kaynak kod üzerinde kullanılması ve benzer kaynak kod tespitinde en iyi sonuçların ortaya konması için olası yöntemlerin her biri uygulanmıştır. Ayrıca bu tez çalışmasında özgün ve yenilikçi yön olarak ve önceki çalışmalardan farklı olarak Word2Vec, Doc2Vec ve derin öğrenme temelli Transformer yöntemleri kullanılmış, Transformer yöntemi ile tüm ölçüm metriklerinde daha başarılı sonuçlar elde edilmiştir.

# 2. TEZİN YAPISI

Bu doktora tezi kapsamında; yazılım geliştirme sürecinde geliştiriciler tarafından eklenen, güncellenen ya da silinen her türlü kod parçası için diğer geliştiriciler tarafından yapılan yorumlardan öğrenerek bir yazılım geliştirici gibi yeni gelecek kaynak kod değişikliklerinde kaynak kod gözden geçirme işlevini sağlayacak sistemlerin inşa edilmesi konusu ele alınmaktadır.

Bu tez çalışması süresince;

- *Kaynak Kod - Gözden Geçirme Yorumu* çiftleri içeren veri seti toplama yazılımı oluşturularak GitHub üzerinde paylaşılmıştır.
- Belirlenen kriterler çerçevesinde *Kaynak Kod - Gözden Geçirme Yorumu* çiftleri içeren ESOGU-CENG-SC veriseti oluşturulmuştur.
- Verilerin vektörlere dönüştürülmeden önce uygulanan çeşitli ön işlemlerin öneri performansını arttırdığı saptanmıştır.
- Literatürde uygulanmış vektöre dönüştürme yöntemlerine ek olarak *Doc2Vec*, *Word2Vec* ve ***Transformer*** yöntemleri ile deneyler yapılarak literatürdeki en iyi yöntemden daha iyi olan *Transformer* yöntemi belirlenmiştir.
- Geçmiş çalışmalardaki deneyler tekrarlanarak ve önerilen yeni yöntemler de dahil edilerek 5'li çapraz doğrulama ile sonuçlar paylaşılmıştır.

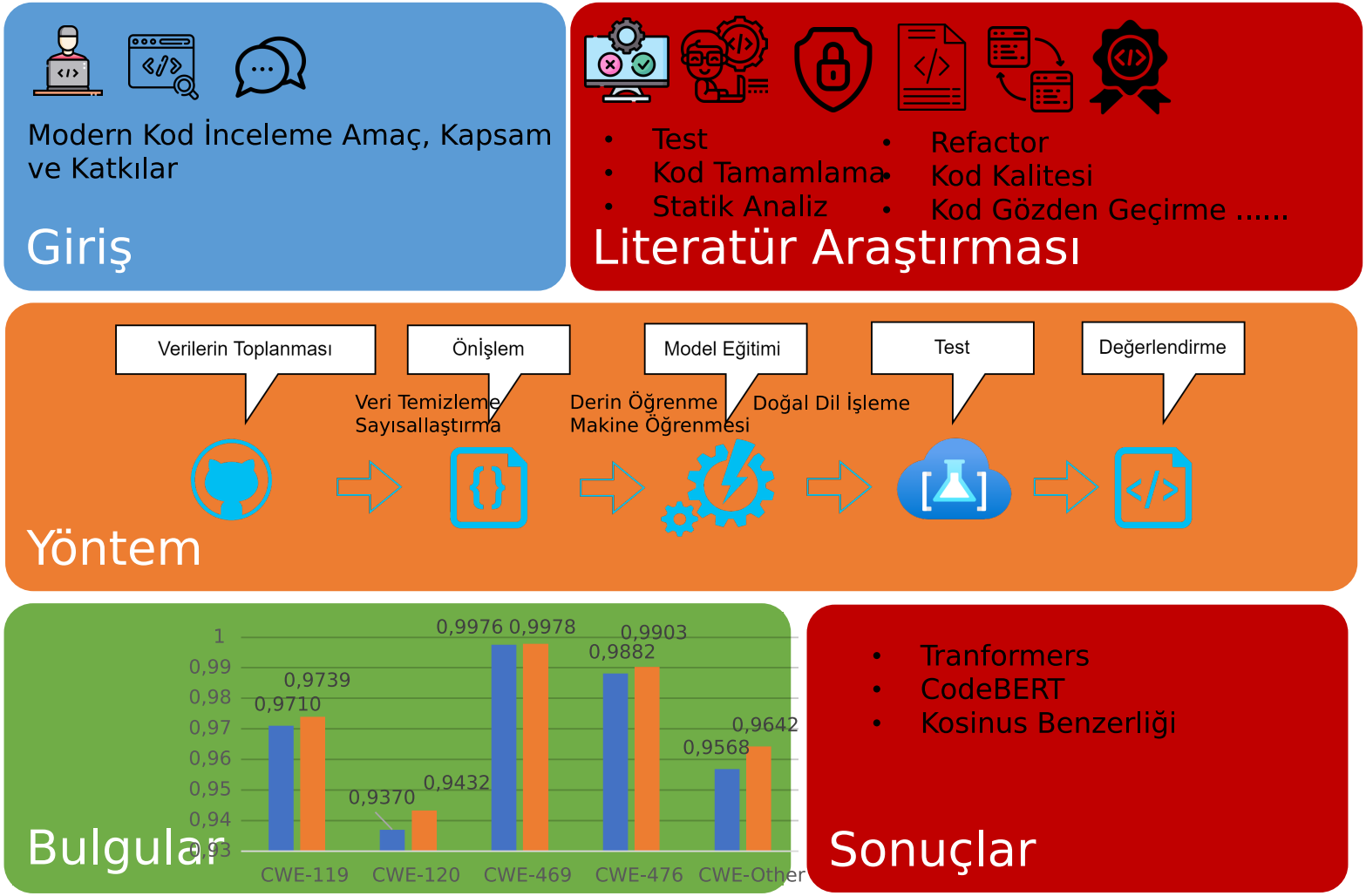
Doktora tezine ait ana başlıklara ve içeriklere özetle Şekil 2.1.'de yer verilmiştir. Tezin sonraki bölümlerdeki ilerleyişi şu şekildedir;

✓ **Bölüm 3**'te girdi olarak kaynak kod kullanan ancak farklı amaçlar için farklı çıktılar üreten literatürdeki farklı alanlar ve bu alanlarda yapılmış önde gelen çalışmalara yer verilmiştir. Bu çalışmalarda kullanılan veri setleri ve uygulanan yöntemler incelenerek kod gözden geçirme için kullanılabilecek yaklaşımlar elde edilmeye çalışılmıştır. Kod gözden geçirme için yapılmış geçmiş çalışmalara bu bölümde detayları ile yer verilmiştir.

✓ **Bölüm 4**'te tez süresince yapılan deneylerde kullanılan veri setleri ile birlikte deneylere girdi olacak verilerin elde edilmesinden çıktıların ölçülmesine

kadar izlenen yöntemler tüm detayları ile yer almaktadır. Bu bölümde geçmiş çalışmalarda uygulanmış yöntemler de dahil olmak üzere önerilen yönteme ait tüm teknikler bu tez kapsamında yeterli ve gerekli detayları ile ele alınmıştır. Her yöntemin uygulandığı adım kendi başlığı altında verilmiştir.

- ✓ **Bölüm 5**'te geliştirilen sistemde önerilen yöntemlerden elde edilen bulgulara önceki çalışmalar ile karşılaştırmalar yapılarak yer verilmiştir. Bu bölümde yapılan karşılaştırmalar literatürdeki en iyi sonuç elde edilen çalışmalar ile yapılmıştır.
- ✓ **Bölüm 6**'da yapılan deneyler ve uygulanan tekniklerin başarıları, avantaj ve dezavantajlarından bahsedilerek gelecekte yapılması olası yaklaşımlara yer verilmiştir.

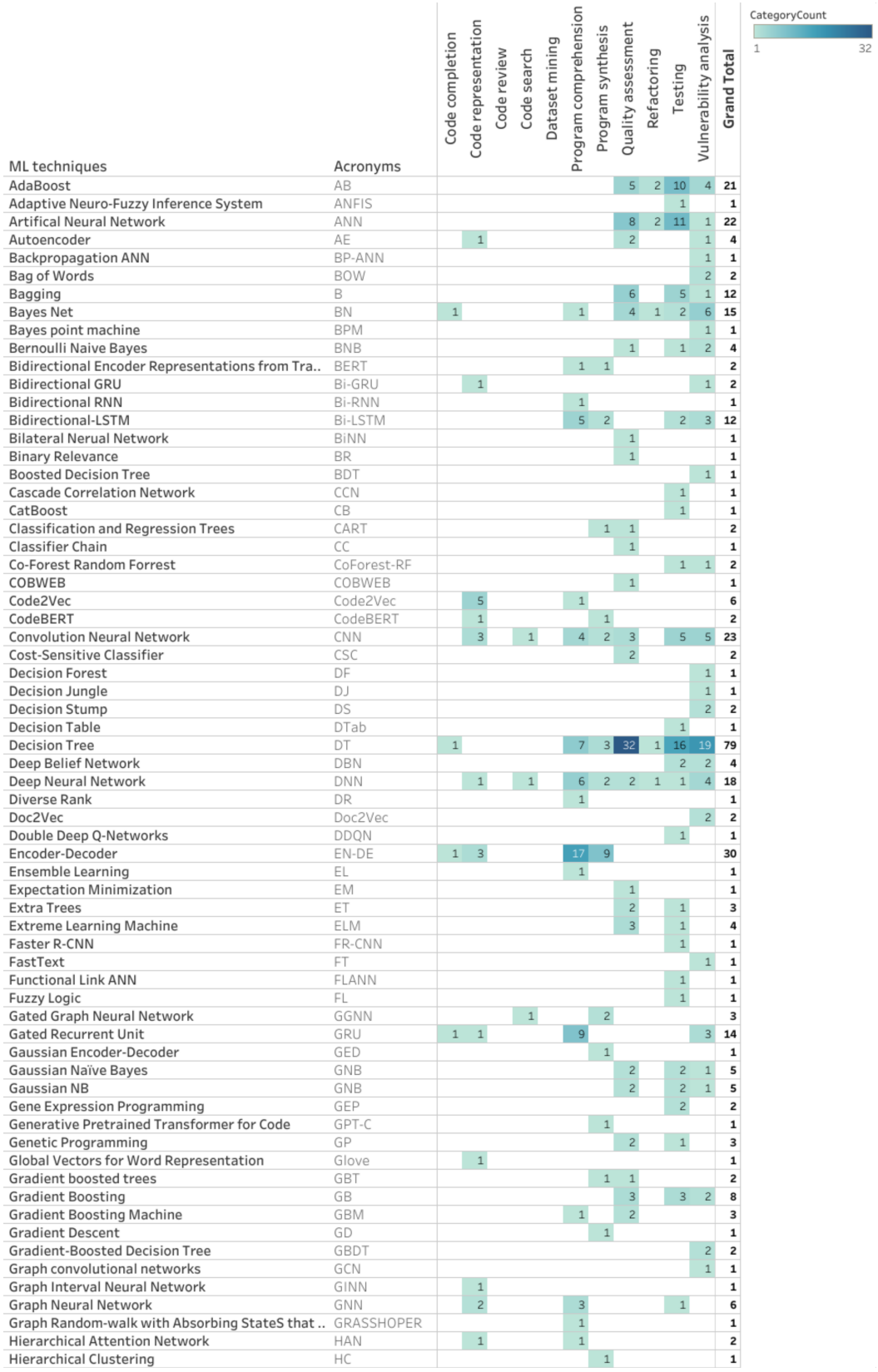



Şekil 2.1. Tezin kapsamı ve içeriği

# 3. LİTERATÜR ARAŞTIRMASI

Bu bölümde, birinci bölümde bahsedilen mevcut süreçler ve konuları doğrudan ya da dolaylı olarak ilgilendiren konular ve çalışmalar ile önerdikleri yöntemlerden bahsedilecektir.

Yazılım güvenilirliğini arttırmak için programlama kurallarının saptanması ve bu kural ihlallerinin hata olarak tespit edilebilmesi için birçok kural tabanlı yöntem önerilmiştir. Bu kurala dayalı yaklaşımlar bir projedeki kuralların anlaşılması için belirli modellerin sıklıkla görülmesine dayanır. Bir desen yeterince sık görülmüyorsa kuralların öğrenilmediği ve bu nedenle birçok hatanın eksik olduğu bilinmektedir. Bu tarz hatalar bir yazılımda geliştirici tarafından bilinmek istenen kötü alışkanlıklar, yazılım hataları veya olağan dışı kullanımlar olabilir (Song Wang vd., 2016). Programlama dilleri de doğal dillere benzer ve hatta kurallara daha bağlı yapılar içermektedir. Doğal dil işleme, insan dilini bilgisayarların anlayabileceği biçimsel bir şekle sokmayı amaçlar. Böylece üzerinde bilgi çıkarma gibi işlemler yapılabilmektedir (Collobert ve Weston, 2008). Doğal dil işleme yöntemleri programlama dilleri içinde benzer şekilde kullanılabilir. Yazılıma ait kaynak kodlar biçimsel bir yapıda olmakla birlikte derin öğrenme, makine öğrenmesi, benzerlik fonksiyonları gibi yöntemler bu yapılardaki benzerliklerin saptanması için kullanılabilir. 2016 yılından bugüne bu öğrenme yöntemlerini çeşitli şekillerde kullanarak yapılan kayda değer bir çok çalışmaya rastlamak mümkündür. Kaynak kodlar üzerinde ihtiyaç duyulan bir çok farklı analiz ve öneri ihtiyaçları bulunmaktadır. Yazılım geliştirme üzerine yapılan çalışmalar makine öğrenmesi ve derin öğrenme tekniklerini ve yaklaşımlarını yazılım testi (J. M. Zhang vd., 2020, Omri ve Sinz, 2020, Lima vd., 2020), kaynak kod temsili (Allamanis, Barr, Devanbu, vd., 2018, Hellendoorn ve Devanbu, 2017), kaynak kod kalitesi (Alsolai ve Roper, 2020, Azeem vd., 2019), program sentezi (Le vd., 2020, Yahav, 2018), kod tamamlama (F. Liu vd., 2020), kod organizasyonu (Aniche vd., 2020), kod özetleme (Z. Liu vd., 2018, LeClair vd., 2019, Allamanis, Barr, Bird, vd., 2015) ve güvenlik açığı tespiti (Ucci vd., 2019, Shen ve S. Chen, 2020, Shabtai vd., 2009) benzeri alanlarda kullanmaktadır (Sharma vd., 2021). Bu alanlarda kullanılan veriler, yöntemler ve çıktılar benzer olsa da üzerinde detaylı çalışma gerektiren farklılıklara sahiptir. Bu çalışma kapsamında odak noktamız modern kod gözden geçirme için bilgisayar bilimleri alanında uygulanmış ve uygulanabilecek yöntem ve yaklaşımları içermektedir.




| ML techniques | Acronyms | Code completion | Code representation | Code review | Code search | Dataset mining | Program comprehension | Program synthesis | Quality assessment | Refactoring | Testing | Vulnerability analysis | Grand Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AdaBoost | AB | | | | | | | | 5 | 2 | 10 | 4 | 21 |
| Adaptive Neuro-Fuzzy Inference System | ANFIS | | | | | | | | | | 1 | | 1 |
| Artifical Neural Network | ANN | | | | | | | | 8 | 2 | 11 | 1 | 22 |
| Autoencoder | AE | | 1 | | | | | | 2 | | | 1 | 4 |
| Backpropagation ANN | BP-ANN | | | | | | | | | | | 1 | 1 |
| Bag of Words | BOW | | | | | | | | | | | 2 | 2 |
| Bagging | B | | | | | | | | 6 | | 5 | 1 | 12 |
| Bayes Net | BN | 1 | | | | | 1 | | 4 | 1 | 2 | 6 | 15 |
| Bayes point machine | BPM | | | | | | | | | | | 1 | 1 |
| Bernoulli Naive Bayes | BNB | | | | | | | | 1 | | 1 | 2 | 4 |
| Bidirectional Encoder Representations from Tra.. | BERT | | | | | | 1 | 1 | | | | | 2 |
| Bidirectional GRU | Bi-GRU | | 1 | | | | | | | | | 1 | 2 |
| Bidirectional RNN | Bi-RNN | | | | | | 1 | | | | | | 1 |
| Bidirectional-LSTM | Bi-LSTM | | | | | | 5 | 2 | | | 2 | 3 | 12 |
| Bilateral Nerual Network | BiNN | | | | | | | | 1 | | | | 1 |
| Binary Relevance | BR | | | | | | | | 1 | | | | 1 |
| Boosted Decision Tree | BDT | | | | | | | | | | | 1 | 1 |
| Cascade Correlation Network | CCN | | | | | | | | | | 1 | | 1 |
| CatBoost | CB | | | | | | | | | | 1 | | 1 |
| Classification and Regression Trees | CART | | | | | | | 1 | 1 | | | | 2 |
| Classifier Chain | CC | | | | | | | | 1 | | | | 1 |
| Co-Forest Random Forrest | CoForest-RF | | | | | | | | | | 1 | 1 | 2 |
| COBWEB | COBWEB | | | | | | | | 1 | | | | 1 |
| Code2Vec | Code2Vec | | 5 | | | | 1 | | | | | | 6 |
| CodeBERT | CodeBERT | | 1 | | | | | 1 | | | | | 2 |
| Convolution Neural Network | CNN | | 3 | | 1 | | 4 | 2 | 3 | | 5 | 5 | 23 |
| Cost-Sensitive Classifier | CSC | | | | | | | | 2 | | | | 2 |
| Decision Forest | DF | | | | | | | | | | | 1 | 1 |
| Decision Jungle | DJ | | | | | | | | | | | 1 | 1 |
| Decision Stump | DS | | | | | | | | | | | 2 | 2 |
| Decision Table | DTab | | | | | | | | | | 1 | | 1 |
| Decision Tree | DT | 1 | | | | | 7 | 3 | 32 | 1 | 16 | 19 | 79 |
| Deep Belief Network | DBN | | | | | | | | | | 2 | 2 | 4 |
| Deep Neural Network | DNN | | 1 | | 1 | | 6 | 2 | 2 | 1 | 1 | 4 | 18 |
| Diverse Rank | DR | | | | | | 1 | | | | | | 1 |
| Doc2Vec | Doc2Vec | | | | | | | | | | | 2 | 2 |
| Double Deep Q-Networks | DDQN | | | | | | | | | | 1 | | 1 |
| Encoder-Decoder | EN-DE | 1 | 3 | | | | 17 | 9 | | | | | 30 |
| Ensemble Learning | EL | | | | | | 1 | | | | | | 1 |
| Expectation Minimization | EM | | | | | | | | 1 | | | | 1 |
| Extra Trees | ET | | | | | | | | 2 | | 1 | | 3 |
| Extreme Learning Machine | ELM | | | | | | | | 3 | | 1 | | 4 |
| Faster R-CNN | FR-CNN | | | | | | | | | | 1 | | 1 |
| FastText | FT | | | | | | | | | | | 1 | 1 |
| Functional Link ANN | FLANN | | | | | | | | | | 1 | | 1 |
| Fuzzy Logic | FL | | | | | | | | | | 1 | | 1 |
| Gated Graph Neural Network | GGNN | | | | 1 | | | 2 | | | | | 3 |
| Gated Recurrent Unit | GRU | 1 | 1 | | | | 9 | | | | | 3 | 14 |
| Gaussian Encoder-Decoder | GED | | | | | | | 1 | | | | | 1 |
| Gaussian Naïve Bayes | GNB | | | | | | | | 2 | | 2 | 1 | 5 |
| Gaussian NB | GNB | | | | | | | | 2 | | 2 | 1 | 5 |
| Gene Expression Programming | GEP | | | | | | | | | | 2 | | 2 |
| Generative Pretrained Transformer for Code | GPT-C | | | | | | | 1 | | | | | 1 |
| Genetic Programming | GP | | | | | | | | 2 | | 1 | | 3 |
| Global Vectors for Word Representation | Glove | | 1 | | | | | | | | | | 1 |
| Gradient boosted trees | GBT | | | | | | | 1 | 1 | | | | 2 |
| Gradient Boosting | GB | | | | | | | | 3 | | 3 | 2 | 8 |
| Gradient Boosting Machine | GBM | | | | | | 1 | | 2 | | | | 3 |
| Gradient Descent | GD | | | | | | | 1 | | | | | 1 |
| Gradient-Boosted Decision Tree | GBDT | | | | | | | | | | | 2 | 2 |
| Graph convolutional networks | GCN | | | | | | | | | | | 1 | 1 |
| Graph Interval Neural Network | GINN | | 1 | | | | | | | | | | 1 |
| Graph Neural Network | GNN | | 2 | | | | 3 | | | | 1 | | 6 |
| Graph Random-walk with Absorbing StateS that .. | GRASSHOPER | | | | | | 1 | | | | | | 1 |
| Hierarchical Attention Network | HAN | | 1 | | | | 1 | | | | | | 2 |
| Hierarchical Clustering | HC | | | | | | | 1 | | | | | 1 |

Şekil 3.1. Kaynak kod üzerine uygulanan makine öğrenmesi teknikleri 1

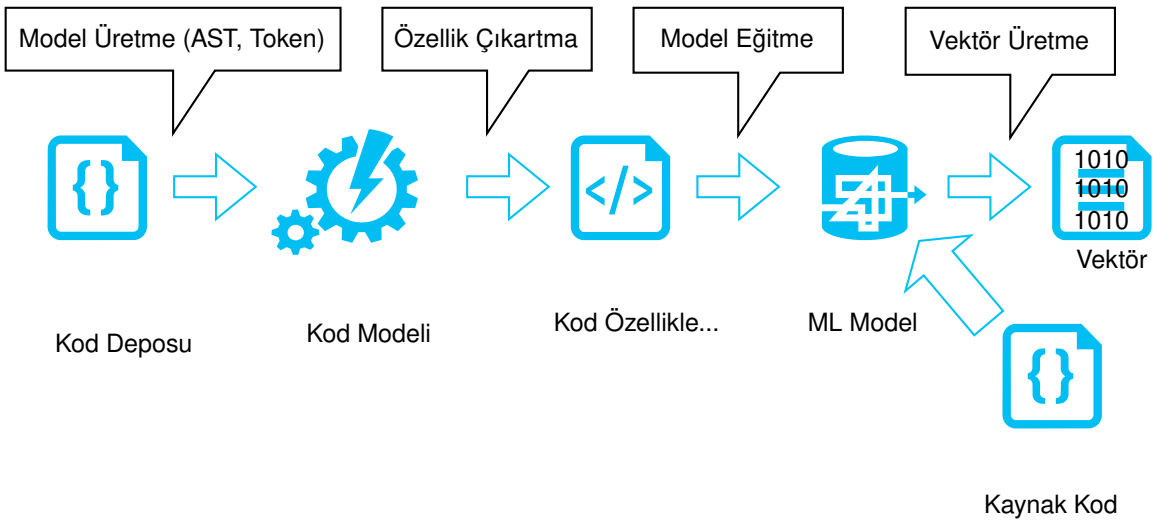
| ML techniques | Acronyms | Code completion | Code representation | Code review | Code search | Dataset mining | Program comprehension | Program synthesis | Quality assessment | Refactoring | Testing | Vulnerability analysis | Grand Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID3 | ID3 | | | | | | | | | | | 1 | 1 |
| JRip | JR | | | | | | | | 2 | | | | 2 |
| K Nearest Neighbours | KNN | | | | | | 3 | | 7 | | 6 | 9 | 25 |
| Kernel based learning | KBL | | | | | | | | | | 1 | | 1 |
| KMeans | KM | | | | | | | | 1 | | | 1 | 2 |
| Kstar | KS | | | | | | | | 4 | | 1 | | 5 |
| Lasso | Lasso | | | | | | | | | | | 1 | 1 |
| Least Median Square Regression | LMSR | | | | | | | | | | 1 | | 1 |
| Light GBM | LGBM | | | | | | | | | | | 1 | 1 |
| Linear Discriminant Analysis | LDA | 1 | 1 | | | | | | 3 | | | 2 | 7 |
| Linear Regression | LR | | | | | | 2 | | 4 | 1 | 6 | 7 | 20 |
| Locally deep support vector machine | LDSVM | | | | | | | | | | | 1 | 1 |
| Logistic Linear Regression | LLR | | | | | | | | | | 1 | | 1 |
| Logistic regression | LOG | 1 | | | | 1 | 1 | 2 | 9 | 4 | 11 | 8 | 37 |
| LogitBoost | LB | | | | | | | | 3 | 1 | 1 | 1 | 6 |
| Long Short Term Memory | LSTM | 4 | 3 | 1 | 1 | | 21 | 9 | 2 | 1 | 5 | 5 | 52 |
| Majority Voting Ensemble | MVE | | | | | | | | 1 | | | | 1 |
| Maximal Marginal Relevance | MMR | | | | | | 1 | | | | | | 1 |
| Modular Tree-structured RNN | MTN | 1 | 1 | | | | | | | | | | 2 |
| Multi Level Perceptron | MLP | | | | | | 2 | 3 | 8 | 1 | 6 | 5 | 25 |
| Multinomial Naive Bayes | MNB | | | | | | | | 2 | 1 | 1 | 1 | 5 |
| Multiple kernel ensemble learning | MKEL | | | | | | | | | | 1 | | 1 |
| Naïve Bayes | NB | | | | | | 7 | 1 | 23 | 2 | 15 | 16 | 64 |
| Neural Language Model | NLM | | | | | | 1 | | | | | | 1 |
| Neural Machine Translation | NMT | 1 | 1 | | | | | 5 | | | 1 | | 8 |
| Neural Network | NN | 1 | | | | | 1 | 1 | | 1 | 1 | 1 | 6 |
| Neural Network for Discrete goal | NND | | | | | | | | | | 2 | | 2 |
| Node2Vec | Node2Vec | | | | | | | | | | 1 | | 1 |
| One Class Classifier | OCC | | | | | | | | 2 | | | | 2 |
| Particle Swarm Optimization | PSO | | | | | | | | | | 1 | | 1 |
| Pointer Network | PN | 1 | | | | | | | | | | | 1 |
| Polynomial regression | POLY | | | | | | | | 1 | | | | 1 |
| Probabilistic Neural Network | PNN | | | | | | | | 2 | 1 | 1 | | 4 |
| Radial Basis Function Network | RBFN | | | | | | | | | | 1 | | 1 |
| Random Forrest | RF | 1 | 1 | | | | 11 | 3 | 27 | 3 | 17 | 21 | 84 |
| RandomTree | RT | | | | | | | | 1 | | 1 | 2 | 4 |
| Recurrent Neural Network | RNN | 3 | 3 | | 1 | | 9 | 5 | 3 | | 3 | 2 | 29 |
| Regression Neural Network | RGNN | | | | | | | | | | 1 | | 1 |
| Reinforcement Learning | RL | | | | | | 3 | | | | | | 3 |
| Residual Neural Network | ResNet | | | | | | | 1 | 1 | | | | 2 |
| Reverse NN | ReNN | | | | | | | | 1 | | | | 1 |
| Ripper | Ripper | | | | | | 1 | | 7 | | 1 | 3 | 12 |
| Sequence-to-Sequence | Seq2Seq | | 1 | | | | 2 | 1 | | | 1 | | 5 |
| Sequential Minimal Optimization | SMO | | | | | | | | 5 | | | 1 | 6 |
| Simple Logistic | SL | | | | | | | | | | | 1 | 1 |
| Statistical Machine Translation | SMT | | | | | | | 1 | | | | | 1 |
| Stochastic Gradient Descent | SGD | | | | | | | | 1 | | 1 | | 2 |
| Support Vector Machine | SVM | | | | | | 8 | 2 | 21 | 4 | 19 | 31 | 85 |
| Support Vector Regression | SVR | | | | | | | | | | 2 | | 2 |
| Text-CNN | Text-CNN | | | | | | | | | | | 1 | 1 |
| Transfer Naïve Bayes | TNB | | | | | | | | | | 1 | | 1 |
| Transformer | TF | 1 | | | | | 5 | | | | 1 | | 7 |
| Word2Vec | Word2Vec | | | | | | | | | | 1 | | 1 |
| XGBoost | XG | | | | | | 2 | 1 | | | 1 | 1 | 5 |

Şekil 3.2. Kaynak kod üzerine uygulanan makine öğrenmesi teknikleri 2

Sharma vd., 2021 tarafından yapılan çalışmada kaynak kod ile ilgili hangi alanlarda hangi makine öğrenmesi tekniklerinin kullanıldığı detaylı olarak ele alınmış, kategori ve alt kategoriler oluşturularak Çizelge 3.1. ve Çizelge 3.2.'de verilmiştir. Bu çizelgeler Sharma vd., 2021 çalışmasından alınmıştır. Kod düzenleme, kod kalitesi, test gibi bazı alanlarda oldukça fazla çalışma olduğu görülmekle birlikte Çizelge 3.1. ve Çizelge 3.2.'de görüleceği üzere kod gözden geçirme üzerine yapılan çalışmalar henüz yeteri kadar ele alınmış değildir.

## 3.1. Kod Temsili (Code Representation)

Bir makine öğrenmesi ya da derin öğrenme modeline kaynak kod ham haliyle doğrudan verilemez. Öncelikle harf, rakam ya da işaretlerden ibaret olan kaynak kodun sayısal gösteriminin elde edilmesi gerekir. Kod temsili, kaynak kodun makine öğrenmesi ya da derin öğrenme modelini besleyebilecek hale getirilmesi için gerekli temel bir faaliyettir (Sharma vd., 2021). Şekil 3.3.'de görüldüğü üzere kaynak kodun sayısallaştırılması için öncelikle Abstract Syntax Tree (AST) (J. Zhang vd., 2019) ya da simge dizileri (Wenhao Zheng vd., 2019) üretecek bir model üretme yöntemi uygulanır. Elde edilen kaynak kod modelinden özellik çıkarma işlemi ile kaynak kod özellikleri çıkarılır. Çıkarılan bu kaynak kod özellikleri makine öğrnemesi ya da derin öğrenme modelinin eğitilmesi için kullanılır. Eğitilen model ile verilen bir kaynak kodun vektör halinde sayısallaştırılmış biçimi elde edilmiş olur ve hata tespiti, sınıflandırma vb. uygulamalar tarafından kullanılabilir.




Şekil 3.3. Kaynak kod sayısallaştırma süreci

## 3.2. Test

Test, bir yazılımın doğruluğunu ve güvenilirliğini artırmak için işlevsel veya işlevsel olmayan hataları belirleme sürecidir. Yazılımlardaki hataları bulmak için makine öğrenmesi yöntemlerinin kullanıldığı çeşitli çalışmalar (Al Qasem vd., 2020, Qiao vd., 2020, Song vd., 2018) bulunmaktadır. Ayrıca makine öğrenmesi teknikleri kullanılarak test senaryosu oluşturma üzerine yapılan çalışmalar da mevcuttur (Braga vd., 2018, Nguyen vd., 2019, Utting vd., 2020).

Kaynak koda ait hata bulma senaryosu için verilerin etiketlenmesi ve özellik çıkarma işlemlerinin ardından makine öğrenmesi modeli oluşturulur. Böylece eğitilmiş makine öğrenmesi modeli ile yeni verilen bir kaynak koda ait hata tespiti yapılabilir. Bir kaynak koda ait hata tespiti yapabilmek için öncelikle eğitim verisinin etiketlenmesi ya da sınıflandırılması gerekmektedir. Bu sebeple bir çok çalışma hazır verisetlerini kullanmış veya kendi verisetlerini oluşturmuşlardır. Çok fazla sayıda çalışma (Al Qasem vd., 2020, Aleem vd., 2015, Bhandari ve R. Gupta, 2018, Butgereit, 2019, Cetiner ve Sahingoz, 2020, Dhamayanthi ve Lavanya, 2019) PROMISE (Sayyad Shirabad ve Menzies, 2005) verisetini kullanırken bazı projeler ek olarak farklı verisetlerini de kullanmışlardır. Hazır verisetlerinin yanı sıra bazı çalışmaların (Pradel ve Sen, 2018) sentetik verisetleri oluşturduğu, bazı çalışmaların (Sotto-Mayor ve Kalech, 2021, Rathore ve S. Kumar, 2021, Fan vd., 2019, Choudhary vd., 2018) ise GitHub'da yer alan Apache, Eclipse, Mozilla gibi bilinen büyük çaplı projelere ait kaynak kodlardan kendi verisetlerini oluşturduğu görülmektedir.

Test senaryosu üretebilecek bir makine öğrenmesi modelinin oluşturulması için genel yaklaşım; bir yazılımdan test edildiği sırada veri yakalama, bu yakalanan verinin işlenerek özellik çıkartılması ve elde edilen özellikler ile makine öğrenmesi modelinin oluşturulması şeklindedir. Verilerin yakalanması için bazı çalışmalar (Braga vd., 2018) uygulamanın hareketlerini kayıt altına alırken bazı çalışmalar (Hu vd., 2018) insan-uygulama etkileşimini kaydederek girdilerin toplanması ve analiz edilmesini sağlamışlardır.

## 3.3. Güvenlik Açığı Analizi (Vulnerability Analysis)

Güvenlik açığı analizi, potansiyel güvenlik açıklarının tespit edilmesi için kaynak kodların analiz edilmesi şeklinde tanımlanabilir. SQL enjeksiyonu, siteler arası komut dosyası çalıştırma (XSS), ortadaki adam saldırısı, şifre saldırıları, DDoS saldırısı gibi çeşitli güvenlik açıklarına sebep olabilecek geliştirmeler kaynak

kodlar içinde yer alabilmektedir. Genel güvenlik açıkları listesi (CWE) olarak tanımlanan ve 1332 zayıflık tipi arasından 459 adet zayıflık tipinin yazılım geliştirme ile ilgili olduğunu belirten MITRE Corporation tarafından yayınlanmış bir liste bulunmaktadır (R. A. Martin ve Barnum, 2008). İlgili ya da yeni zayıflıkların güncellenmesi devam eden bir süreç olmakla birlikte çeşitli zamanlarda derlenerek yayınlanmıştır (B. Martin, Brown, Paller, Kirby ve Christey, 2011).

Bir kaynak kod parçasında karşılaşılabilecek güvenlik açıklarının tespit edilmesini ya da sınıflandırılmasını sağlamak üzere yapılan makine öğrenmesi yöntemleri ve çalışmaları (Du vd., 2019, Cui vd., 2020, Aakanshi Gupta vd., 2021) bulunmaktadır. Kendi verisetlerini oluşturmayı tercih eden çalışmalar (Aakanshi Gupta vd., 2021, Piskachev vd., 2019, Cui vd., 2020, Wei Zheng vd., 2020) olduğu gibi etiketlenmiş hazır verisetlerini kullanan çalışmalar (Y. Zhang ve B. Li, 2020, Yosifova vd., 2021, Yang vd., 2019) da mevcuttur. Güvenlik açığı analizi ile ilgili çalışmaların çoğunluğunun güvenlik açığı analizini öncelikle bir sınıflandırma problemi olarak ele aldığı görülmektedir.

Mevcut çalışmalar bir kaynak kod parçasındaki güvenlik açığını tespit edebilmek için oluşturulan makine öğrenmesi modelinin eğitimi için öncelikle hazır bir veriseti kullanmış ya da bir veriseti oluşturmuştur. Daha sonra elde edilen bu verisetini kullanarak özellik çıkarılmış ve ardından da elde edilen bu özellikler ile makine öğrenmesi modeli eğitilmiştir. Böylece artık elde edilen eğitilmiş model ile yeni bir kod parçası için tahminlerde bulunulabilmektedir.

## 3.4. Program Sentezi (Program Synthesis)

Bu başlık altında genel olarak otomatik hata düzeltme, kod taşıma gibi farklı yaklaşımlar tartışılmaktadır. Otomatik hata düzeltme, bir uygulama içinde bulunan belirli bir yazılımsal hataya karşılık otomatik olarak düzeltme kodunun önerilmesi şeklinde tanımlanabilir. Kod taşıma ise bir programlama dilinde yazılmış kaynak kodların yapısal olarak analiz edilerek başka bir programlama dilinde yeniden yazılması işleminin otomatize edilmesidir.

Otomatik hata düzeltme genel olarak 2 aşamadan oluşmaktadır. Öncelikle hatanın nerede olduğunun belirlenmesi sağlanır ardından tespit edien hata için düzeltme önerilir (Sharma vd., 2021). Birçok çalışma hataların tespiti için test senaryolarını kullanmaktadır. Bazı test senaryolarının başarısız olması ilgili kod

parçalarında bir sorun olduğu anlamına gelmektedir ama bazı test senaryoları ise başarılı sonuçlar ile kaynak kodlarda değişiklik olmaması gerektiğini söyler. Bu durumda hatanın nerede olduğunu tespit etmek olasıdır ve amaç tüm test senaryolarının başarılı olduğu bir çıktı elde etmektir. Böylece önerilen kod düzeltmesi tüm testlerin başarılı bir şekilde geçmesini sağlayabilir (Goues vd., 2019). Otomatik hata düzeltmesine odaklanan çalışmalar kaynak kod ve bu kodlar üzerinde hata düzeltmeleri içeren bir veriseti kullanmakta ya da oluşturmaktadırlar. Elde edilen bu veriseti üzerinde özellik çıkarma işlemi yapılır. Elde edilen özellikler ile makine öğrenmesi modeli eğitilir ve böylece gelecek bir kaynak kod için tahminlerde bulunabilecek eğitilmiş bir model elde edilmiş olunur (Sharma vd., 2021).

Kod taşıma için yapılan çalışmalar farklı teknikler sunmaktadır. Örneğin, Le vd., 2020 makine çevirisi algoritmaları ve uygulamalarını da kapsayan derin öğrenme teknikleri üzerine bir çalışma yapmıştır (Le vd., 2020).

## 3.5. Kod Arama (Code Search)

Kod arama, kullanıcıların ihtiyaçlarına yönelik kod parçalarını bulma sürecidir. Bu süreçte, kaynak kodu ve açıklamaları içeren bir eğitim seti hazırlanır. Ardından, giriş kodundan ve metinden ilgili özellikler çıkarılır. Bu özellikler, makine öğrenimi modelleriyle eğitilir ve test sorgularını yürütmek için kullanılır. Eğitim sonrasında, bir sorgu ve kod parçası kullanılarak kod araması gerçekleştirilebilir. Bu süreçte, anlamsal benzerlik ölçümü ve sıralama yöntemleri kullanılır. Kod arama çalışmaları, genellikle gömme (embedding) ve dikkat mekanizmaları (Attention Mechanism) gibi teknikler kullanarak kod parçalarının temsillerini öğrenir ve benzerlikleri değerlendirir (Sharma vd., 2021).

Kod arama çalışmalarında çeşitli yöntemler kullanılmıştır. Kod arama çalışmalarında farklı yaklaşımların ve tekniklerin kullanıldığını şu şekilde özetleyebiliriz;

- Veri Seti Hazırlama: Çalışmalarda, kaynak kodu ve açıklamaları veya sorguları içeren bir eğitim seti hazırlamak için farklı teknikler kullanılmıştır. Bazı çalışmalarda, açıklamalı kodlar kullanılmıştır. Diğerleri ise kaynak kodunu tokenlar, soyut sözdizim ağaçları (Abstract Syntax Tree (AST)) ve kontrol akış grafikleri (CFG) şeklinde temsil etmiştir. Ayrıca, GitHub gibi platformlardan yazılım projeleri çekilerek veri seti oluşturulmuştur.

- Özellik Çıkarımı: Kod arama çalışmalarında, giriş kodundan ilgili özelliklerin çıkarılması önemlidir. Bu amaçla, gömme (embedding) yöntemleri yaygın olarak kullanılmıştır. Örneğin, kodun farklı bileşenleri (metot adı, API sıralaması, tokenlar) ayrı ayrı işlenerek gömme vektörleri elde edilmiştir. Bazı çalışmalarda ise kod açıklamaları için ayrı gömme vektörleri oluşturulmuştur. Ayrıca; Long Short-Term Memory (LSTM), Tree Long Short-Term Memory (Tree-LSTM) ve Graph Neural Network (GNN) gibi yapılar kullanılarak çoklu modalitelerin temsili öğrenilmiştir.
- Makine Öğrenimi Modeli Eğitimi: Çalışmalarda farklı makine öğrenimi modelleri kullanılmıştır. Örneğin, Evrişimsel Sinir Ağı (Convolutional Neural Network (CNN)) tabanlı bir model, gömülü kod ve sorgu arasındaki bağımlılıkları öğrenmek için bir eş zamanlı dikkat mekanizması kullanmıştır. Bir başka çalışmada ise çoklu modalitelerin dikkat birleştirme yöntemi kullanılarak temsilleri öğrenilmiştir. Ayrıca, kelime ve belge gömme kullanarak kod araması yapılan modeller ve otomatik kodlayıcı ağları da kullanılmıştır.
- Kod Arama: Eğitilmiş makine öğrenimi modelleri kullanılarak kod araması gerçekleştirilmiştir. Bu adımda, verilen sorgu ve kod parçaları arasındaki benzerlik ölçülmüş ve sıralama yapılmıştır. Örneğin, benzerlik ölçümü için kosinüs benzerliği kullanılmış ve kod parçaları giriş sorgusuna göre sıralanmıştır (Shuai vd., 2020).

## 3.6. Kod Tamamlama (Code Completion)

Kod tamamlama, modern kaynak kod düzenleyicilerinin ve geliştirme ortamlarının vazgeçilmez bir özelliğidir. Bu alanda yapılan çalışmalar genellikle geniş veri kümelerinden madencilik yaparak kendi veri setlerini oluşturmuşlardır (Gopalakrishnan vd., 2017). Bu veri setleri kullanılarak dil işleme ve makine öğrenimi modelleri eğitilir. Özellik çıkarımı aşamasında, kaynak kodunun çeşitli formlarından yapısal ve sözdizimsel bilgiler çıkarılır. Eğitilmiş modeller, kod tamamlama için sorgu bağlamına uygun öneriler sunar. Bu çalışmalarda, Recurrent Neural Network (RNN), olasılıksal modeller ve çoklu görev öğrenme gibi çeşitli makine öğrenimi teknikleri kullanılır (Shuai Wang vd., 2019). Ayrıca, dil modelleri, dikkat mekanizmaları ve istatistiksel öğrenme teknikleri de uygulanır (Zhong vd., 2019). Bu çalışmalar, geliştiricilerin kod tamamlama sürecini desteklemek ve yazılım anlama faaliyetlerini geliştirmek amacıyla gerçekleştirilir.

## 3.7. Kod Yeniden Düzenleme (Refactoring)

Bu bölümde, kaynak kod analizi ve makine öğrenimi tekniklerinin uygulanmasıyla refactoring adaylarının belirlenmesi veya refactoring işlemlerinin tahmin edilmesi için yapılan çalışmalar özetlenmektedir. Çalışmalar genellikle üç aşamalı bir süreç kullanmaktadır. İlk aşamada, projelerin kaynak kodu kullanılarak eğitim için bir veri seti hazırlanır. Ardından, bireysel örneklerden (yani bir yöntem, sınıf veya dosya) ilgili özellikler çıkarılır. Çıkarılan özellikler, bir makine öğrenimi modeline beslenerek eğitilir. Eğitildikten sonra, model girdi bir örneğin refactoring için aday olup olmadığını tahmin etmek için kullanılır.

Bu çalışmaların birçoğu kendi model eğitimi için kendi veri setlerini oluşturmuştur. Bazı çalışmalar kod kalite metriklerine dayanarak özellikler çıkarmıştır. Diğer çalışmalar ise kod kalitesi metriklerinin yanı sıra süreç metrikleri ve kod sahipliği metriklerini kullanmıştır. Makine öğrenimi modelleri olarak çeşitli algoritmalar kullanılmıştır, örneğin Naive Bayes, Destek Vektör Makinesi, Rastgele Orman gibi algoritmalar tercih edilmiştir (Kosker vd., 2009; L. Kumar ve Sureka, 2017).

Özetle, bu çalışmalar refactoring adaylarını belirlemek veya refactoring işlemlerini tahmin etmek için kaynak kod analizi ve makine öğrenimi tekniklerini kullanmaktadır. Farklı özellik çıkarım yöntemleri ve makine öğrenimi modelleri kullanılarak, kod kalitesini iyileştirmek ve program davranışını korurken refactoring sürecini otomatikleştirmek amaçlanmaktadır.

## 3.8. Kod Kalitesi (Code Quality)

Bu kategorideki çalışmalar, güvenilirlik, sürdürülebilirlik ve çalışma zamanı performansı gibi çeşitli kalite özellikleri ile ilgili sorunları değerlendirmekte veya tahmin etmektedir. Süreç, veri seti ön işleme ve etiketleme ile başlarak etiketlenmiş veri örnekleri elde edilir. İşlenmiş örnekler üzerinde özellik çıkarım teknikleri uygulanır. Çıkarılan özellikler daha sonra bir makine öğrenimi modeline beslenerek eğitilir. Eğitilmiş model, analiz edilen kaynak kodunda kalite sorunlarını değerlendirir veya tahmin eder.

Veri seti ön işleme ve etiketleme aşamasında, çalışmalar farklı yaklaşımlar kullanmıştır. Özellik çıkarımında ise bazı çalışmalar kaynak kodu metriklerini kullanırken, diğerleri düşük düzeyde kod özelliklerini veya uyarıları çıkararak özellikler oluşturmuştur. Makine öğrenimi modeli eğitiminde çeşitli algoritmalar

kullanılmıştır, örneğin Destek Vektör Makinesi, Rastgele Orman, K En Yakın Komşu ve Karar Ağacı gibi algoritmalar tercih edilmiştir (Amorim vd., 2015; Kaur vd., 2017).

Özetle, bu çalışmalar kaynak kodu analizi için makine öğrenimi tekniklerini kullanarak kalite sorunlarını değerlendirmeyi veya tahmin etmeyi amaçlamaktadır. Veri seti hazırlama, özellik çıkarımı ve makine öğrenimi modeli eğitimi adımlarını içeren süreçler kullanılarak, statik analiz ve kalite özelliklerine odaklanarak yazılım geliştirme sürecinin iyileştirilmesi hedeflenmektedir.

## 3.9. Kod Gözden Geçirme (Code Review)

Kod gözden geçirme diğer bir deyişle kod inceleme, bir geliştirici tarafından yazılan kodun bir veya daha fazla farklı geliştirici tarafından sistematik olarak kontrol edilmesi işlemidir. Bu başlık altında geliştiriciler tarafından yazılım geliştirme süreci rutini içerisinde yapılan kod gözden geçirme eyleminin makine öğrenmesi veya derin öğrenme gibi yöntemlerle desteklenmesi üzerine yapılan çalışmalar inceleme altına alınmıştır.

Eğitim aşamasında ya da yazılım geliştirme aşamasında geliştiriciye henüz farkında olmadığı ya da diğer tecrübeli geliştiriciler tarafından zaten tespit edilmiş eksik ya da hatalı kod parçalarını tespit ederek geçmiş yorumlar çerçevesinde kaynak koddaki hatanın düzeltilmesine destek olacak bir sistem daha kaliteli bir kod geliştirme süreci sunacaktır. Bu tez kapsamında geliştirilen sistem ile derin öğrenme tabanlı sistemlere göre hem yeniden eğitim maliyetleri açısından hem de yaptığı öneriler açısından daha verimli ve daha başarılı sonuçlar elde edilmiştir. Vektörlere dönüştürme işlemlerinde kullanılan transformers yöntemi sonuçlarda fark edilir bir etki yaratmıştır. Böylece yapılan önerinin doğruluğunun kaynak kodun sayısallaştırılma biçimiyle büyük ölçüde ilişikili olduğu görülmektedir.

Lal ve Pahwa tarafından yapılan çalışmada kod örnekleri öncelikle hatalı veya uygun şekilde işaretlenmiştir. Bu kod örnekleri üzerinde, örnekleri vektörlere dönüştürmek için TF-IDF kullanmadan önce normalleştirme ve etiket kodlama gibi kapsamlı ön işlemler gerçekleştirilmiştir. Kod örneklerini hatalı ya da temiz olarak sınıflandırmak için bir Naive Bayes modeli kullanılmıştır (Lal ve Pahwa, 2017).

Axelsson vd., gözden geçirenlerin sorunlu kod bölümlerini bulmalarına yardımcı olmak için "Code Distance Visualise (Kod Mesafesi Görüntüleyici)" olarak

adlandırdıkları bir araç geliştirmişlerdir. Araştırmacılar, deneylerini açık kaynak ve özel kod depolarında ayrı ayrı test etmişlerdir. Burada en yakın k komşuluğuna (kNN) dayanan (NCD) metriğine dayalı, etkileşimli, denetimli, kendi kendine öğrenen bir statik analiz aracı sunmuşlardır (Axelsson vd., 2009).

White vd., 2016 yılında derin öğrenme teknikleri kullanarak Java kaynak kod depoları üzerinde yaptıkları çalışmada geleneksel yöntemlere göre öğrenme tabanlı yaklaşımın kaynak kod üzerinde benzerlik gösteren kısımların daha doğru oranlarla tespit edildiğinden bahsetmişlerdir (White, M. Tufano, Vendome, vd., 2016). Aynı çalışma grubu tarafından yapılan kaynak kod merkezli diğer çalışmalarda derin öğrenme tabanlı kod benzerlikleri ile kod onarma çalışması, hata düzeltmeler ile kaynak kod mutasyonu gibi konular araştırılmıştır (M. Tufano, Watson, Bavota, Di Penta, vd., 2018, M. Tufano, Watson, Bavota, Di Penta, vd., 2019). Kod onarma için yapılan çalışmada karşılaştırmalı deneyler 374 yazılım hatası revizyonu içeren açık kaynaklı bir Java projesi üzerinde yapılmıştır. Sonuçlar öğrenme tabanlı bu yaklaşımın diğer yöntemlerde bulunamayan hataları tespit ettiğini göstermektedir (White, M. Tufano, Martinez, vd., 2019).

Harer vd. 2018 yılında C/C++ kodları üzerinde yaptıkları çalışmada 2 farklı yaklaşım üzerinde durmuşlardır. İlki derleme ve inşa etme aşamalarından çıkarılan özellikleri kullanırken ikincisi doğrudan kaynak koddan elde edilen özellikleri kullanmaktadır. Bu iki yaklaşım birbirinden farklı özellikler vermektedir. Kaynak kod üzerinde yazılımın nasıl kodlandığı bilgisine erişilebilirken diğer yaklaşım ile derleyiciye özel durumların ortaya konması amaçlanmıştır. Çeşitli makine öğrenme tekniklerinin statik analiz araçlarının çıktılarını öngörmede başarılı olduğu ortaya konmuştur. Bu tarz yaklaşımlar ile geliştiricilerin kod gözden geçirme ve daha hatasız yazılımlar üretme konusunda daha başarılı olabileceklerini belirtmişlerdir (Harer vd., 2018).

Gupta ve Sundaresan yaptıkları çalışmada otomatik, esnek ve adaptif bir kod gözden geçirme sistemi önermişlerdir. Böylece kaynak kodlarda tespit edilen benzer problemler için nasıl öneride bulunacağını öğrenen bir sistem tasarlamışlardır. Derin öğrenme tekniklerinin kullanıldığı sistem, geçmiş kod gözden geçirme çıktılarını referans alarak kodlar üzerinde kod gözden geçirme yapmak üzere tasarlanmıştır. Araştırmacılar tarafından yayınlanmamış kod depoları üzerinde yapılan çalışmada başarıyı ölçmek için anket düzenlemişler ve test sonuçlarının başarılı olduğunu belirtmişlerdir (Anshul Gupta ve Sundaresan, 2018).

Tufano vd. iki farklı derin öğrenme mimarisi ile hem kod geliştiriciye hem de gözden geçirene önerilerde bulunabilecek bir sistem tasarlamayı hedeflemişlerdir. Bu mimariler, koddan koda ve koddan yoruma doğru eşleşmeler üretmek için tasarlanmıştır. Geliştirdikleri sistemde dikkat çekici sonuçlar elde etmelerine rağmen tam anlamıyla işlevsel hale gelmeleri için daha fazla çalışma yapılması gerektiğini belirterek çalışmalarını sonlandırmışlardır (R. Tufano, Pascarella, vd., 2021). Ardından, yaklaşımlarını değiştirerek, ikinci bir çalışmada 'Metin-Metin Dönüştürücüleri' kullanan yeni bir sistem önermişlerdir. Önerilen sistemin derin öğrenme (R. Tufano, Pascarella, vd., 2021) ile uygulanan yaklaşımdan daha verimli olduğunu savunmuşlardır. Tufano vd. veri setini önemli ölçüde genişlettiklerini ve veri setini bir ön işlemden geçirerek ve ince ayar yaparak daha başarılı bir model ortaya çıkardıklarını belirtmişlerdir. Hem önceden eğitilmiş hem de önceden eğitilmemiş modeller ile test ederek modelin ön eğitiminin oldukça etkili olduğundan bahsetmişlerdir (R. Tufano, Masiero, vd., 2022).

Siow vd. kod değişikliklerine ve ilgili incelemelere yani kod gözden geçirmelere dayalı çalışmalarında statik analiz veya derin öğrenmeye dayalı yaklaşımlardan daha iyi performans göstermeyi amaçlamışlardır. Bu nedenle kullandıkları derin öğrenme modeline metin gömmelerini ekleyerek daha başarılı sonuçlar elde ettiklerini belirtmişlerdir (Siow vd., 2020).

Hong ve Tantithamthavorn, CommentFinder adlı bir sistem önermişlerdir. Kelime çantası yöntemi kullanılarak kod bloklarının vektörleştirilmesine ve kosinüs benzerliği yöntemi kullanılarak bu vektörlerin yakınlığının ölçülerek kod bloklarının benzerliğinin ölçülmesine dayalı bir yaklaşımdır. Ek olarak, en iyi 1 kod inceleme yorumunu önerirken tam eşleşme için önceki çalışmadan %32 daha iyi sonuç elde ettiklerini belirtmişlerdir. Önerdikleri yöntem yalnızca daha iyi doğruluk sağlamakla kalmayarak aynı zamanda önceki çalışmaya göre 49 kat daha hızlı çalışmaktadır (Hong vd., 2022).

Li vd. "CodeReviewer" adını verdikleri önceden eğitilmiş bir derin öğrenme modeli önermişlerdir. 3 aşama olarak tasarlanan yaklaşımın ilk aşamasında, değişikliğin yüksek kalitede olup olmadığı ikili (0,1) olarak verilerek tespit edilmek istenmiş ve yüksek kaliteli değişikliklerin gözden geçirmeye hazır olduğu kabul edilmiştir. İkinci adımda, kaynak koddaki değişiklik ve yorumlara dayalı bir yaklaşımla bir kod değişikliği için bulunan en iyi eşleşmeleri önermeye odaklanmışlardır. Önerilen eşleşmelerden uygun olanını seçme görevini geliştiricinin kendisine bırakmışlardır. Üçüncü aşamada ise geliştiriciye kod üzerinde yapılan yorumlara göre kodu nasıl yeniden düzenleyeceğini gösteren bir

sistem önermişlerdir. Bulgularına göre, önerilen bu modelin önceki derin öğrenmeye dayalı en iyi yaklaşımlardan daha iyi performans gösterdiğini savunmuşlardır (Z. Li vd., 2022).

Li vd. metinden metne T5 modelini kullanarak kod inceleme yorumları üreten "AUGER" adlı bir sistem tasarlamışlardır (L. Li vd., 2022). Tufano vd. (R. Tufano, Masiero, vd., 2022) tarafından yapılan ilk çalışma ile karşılaştırıldığında ROUGE-L puanında %37,38'lik bir artış elde ettiklerini belirtmişlerdir. Bu çalışmanın performansını artıran unsurlar arasında; verilerin ön işleme alınması, kodun incelenen bölümünün etiketlenmesi, kod ile gözden geçirme yorumu arasındaki ilişkiyi daha açık hale getirmek için çapraz öneğitim ve maskeleme olduğundan bahsetmişlerdir (L. Li vd., 2022). Bu çalışma, kod inceleme yorumlarının üretilmesinde veri adaptasyonunun önemini göstermektedir.

Kod gözden geçirme süreçlerinin otomatikleştirilmesine ek olarak, kaynak kodlar üzerinde önceki başlıklarda özetle verilen alt görevler üzerinde de birçok çalışma yapılmıştır. Bunlardan en öne çıkan çalışma, transformatör (transformers) tabanlı bir sistem olan "CodeBERT" (Z. Feng vd., 2020)'dir. Bu çalışmanın temel amacı, doğal dil ve kaynak kod arasındaki anlamsal ilişkiyi ortaya koymaktır. Kaynak kod dokümantasyonu üretilmesi gibi kod ve doğal dilin birlikte kullanıldığı görevlerde kullanılmıştır. Kaynak kod içerisinde yer alan doğal dil işlenmesi için model oluşturulması noktasında önemli bir kilometre taşıdır.

Lu vd. CodeXGlue (Lu vd., 2021) isimli çalışmada, kaynak kod anlama ve üretme ile ilgili görevlerde kullanmak üzere bir veri seti oluşturmuşlardır. Bu çalışmada hedeflenen görev olan klonlanan kaynak kodların tespit edilmesi için "CodeBERT" modeli kullanılmıştır. Ayrıca; kod tamamlama, kod özetleme ve benzeri birçok görev için farklı modellerde küçük ayarlamalar yapılarak çalışmalar sunulmuştur.

Kodların vektörel gösterimleri üzerine bu tez kapsamında da yer verildiği üzere farklı çalışmalar mevcuttur. Chen ve Monperrus (Z. Chen ve Monperrus, 2019) tarafından yapılan literatür taraması, kod gömme ile ilgili çalışmaların kapsamlı bir incelemesini sunmaktadır. Ayrıca CodeBERT'ten önce Kanade vd. "BERT" modelini kaynak kod metinleriyle önceden eğiterek "CuBERT" (Kanade vd., 2020) adlı bağlamsal bir kod temsil modeli geliştirmişlerdir. Ayrıca transformasyon modeli olarak da kullanılan UniXCoder (Guo vd., 2022) içerisinde Guo vd. kaynak kod metinlerinin temsili için çok katmanlı bir dönüştürücü

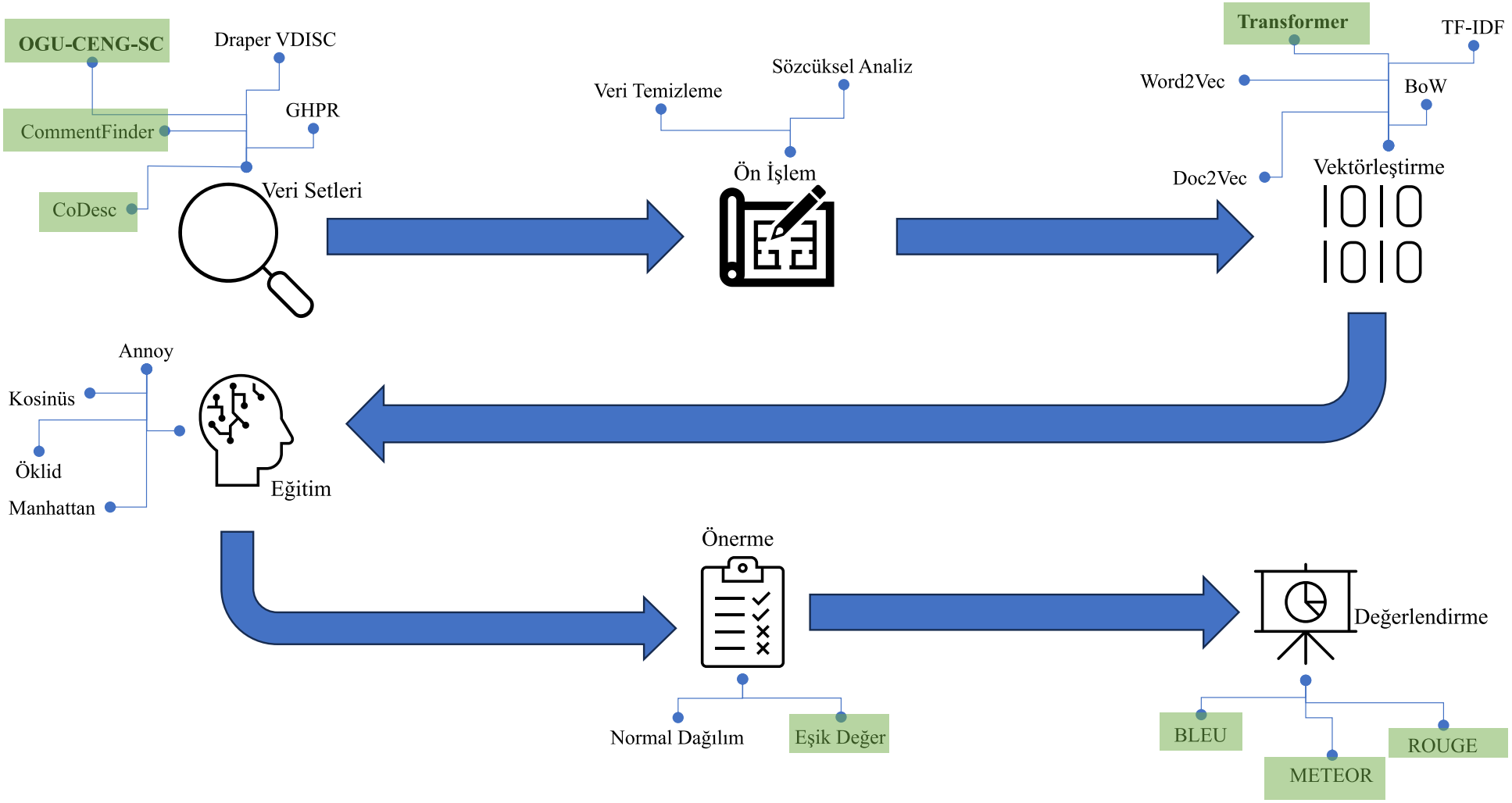
mimarisi kullanmışlar ve kod metinlerini AST ayrıştırma yöntemi yoluyla parçalayarak sayısallaştırmışlardır.

# 4. YÖNTEM

Tez çalışması süresince kaynak kodlar üzerinde yapılan çalışmaların yanı sıra doğal dil işleme ile ilgili çalışmalardan yola çıkarak uygulanabilecek yöntemler de incelenmiştir. İncelemeye alınan önceki çalışmalarda ortaya konan deneysel çalışmalar tekrar yapılmıştır. Gerek açık kaynak kod depolarında yayınlanan ve topluluklar tarafından desteklenen projeler üzerinde gerekse de başka çalışmalar tarafından oluşturulmuş gerçek ya da sentetik veri setleri üzerinde çeşitli yöntemler uygulanarak elde edilen sonuçların doğruluğu kesinleştirilmeye çalışılmış ve geçmiş çalışmalara olan katkılar sunulmaya çalışılmıştır.

Verilerin toplanması ya da veri setlerinden elde edilmesinin ardından girdi olarak sağlanan veriler çeşitli şekillerde ve adımlarda ön işlemlerden geçirilerek verilerin filtrelenmesi, temizlenmesi ve bazı durumlarda ilgili kelime ya da kelime öbeklerini temsil edecek etiketler ile işaretlenmesi sağlanmıştır. Elde edilen yeni girdiler ve bu girdilere karşılık gelen çıktılar, modellerin eğitimi için kullanılarak oluşturulan modeller ile test verileri üzerinde tahmin çalışmaları yapılmıştır. Test verileri ile elde edilen çıktıların doğruluğu da doğal dil işleme alanında sıklıkla kullanılan BLEU, ROUGE, METEOR gibi çeşitli geçerleme yöntemleri ile ölçülmüştür (Şekil 4.1.). Bir modelin akışına ait adımlarda özellikle literatüre ilk defa sunulan ya da etkili görülen adımlar Şekil 4.1. içerisinde yeşil zemin ile verilmiştir.

Tezin bu bölümünde tez çalışması sürecinde kullanılan yöntemler ve teknikler ile geçmiş çalışmalarda kullanılan bazı yöntemler özetler halinde verilmektedir.




Şekil 4.1. Uygulanan yöntemlerin akışı

## 4.1. Veri Setleri

Bu bölümde kaynak kod içeren, bu tez kapsamında mercek altına alınan ve kullanılan veri setleri özetlenmektedir. GitHub kod deposu üzerinden elde ettiğimiz ve bu tez kapsamında oluşturduğumuz verisetinin oluşturulması ve içeriği hakkında da detaylı bilgiye yer verilmektedir.

### 4.1.1. Draper VDISC veri seti

Draper VDISC, açık kaynak projelerden toplanmış 1,27 milyon fonksiyon içeren bir veri setidir. Fonksiyonlar, statik analiz yöntemiyle potansiyel güvenlik açıkları tespit edilerek Ortak Güvenlik Açığı Listesi'nde (Common Weakness Enumaration (CWE)) yer alan 5 güvenlik açığı ile etiketlenmiştir. Bu güvenlik açıkları CWE-120, CWE-119, CWE-469, CWE-476 ve CWE-other şeklindedir. Veri setinde yer alan her bir sınıf etiketine ait veri sayılarına ait detaylar Çizelge 4.1.'de verilmiştir. Veri seti C ve C++ programlama dillerinde yazılmış fonksiyonlar ve bu fonksiyonlarda bulunan güvenlik açıklarına ait etiketlemeleri içermektedir. Bir fonksiyon birden fazla güvenlik açığı barındırabileceği gibi hiç bir güvenlik açığı da barındırmayabilmektedir (R. Russell vd., 2018).

Çizelge 4.1. Draper VDISC veri seti sayısal bilgiler

| | Güvenlik Açığı | CWE-119 | CWE-120 | CWE-469 | CWE-476 | CWE-other |
|---|---|---|---|---|---|---|
| Eğitim (Train) | Var | 19286 | 38019 | 2095 | 9694 | 27959 |
| | Yok | 1000185 | 981452 | 1017376 | 100977 | 991512 |
| Doğrulama (Validation) | Var | 2419 | 4750 | 252 | 1208 | 3579 |
| | Yok | 125057 | 122726 | 127224 | 126268 | 123897 |
| Test | Var | 2452 | 4891 | 278 | 1192 | 3490 |
| | Yok | 124967 | 122528 | 127141 | 126227 | 123929 |

CWE, yazılım ve donanım zayıflık türlerinin bir listesidir (B. Martin, Brown, Paller, Kirby ve Steve Christey, 2011). Veri setinde yer alan etiket bilgileri Çizelge 4.2.'de yer almaktadır. Bu veri seti eğitim (%80), doğrulama (%10), test (%10) olarak üç parçaya ayrılmış durumdadır. Bu şekilde ele alındığında veri setinde yer alan toplam bölünmüş kayıt sayıları Çizelge 4.3. ile verilmiştir.

Çizelge 4.2. Draper VDISC veri seti etiketleri

| | |
|---|---|
| CWE-119 | Bir Bellek Arabelleğinin Sınırları İçinde İşlemlerin Uygunsuz Kısıtlanması |
| CWE-120 | Giriş Boyutunu Kontrol Etmeden Arabellek Kopyalama |
| CWE-469 | Boyutu Belirlemek için İşaretçi Çıkarma Kullanımı |
| CWE-476 | NULL Pointer Referansı |
| CWE-Other | CWE-119, CWE-120, CWE-469 ve CWE-476 dışındaki hatalar |

Çizelge 4.3. Draper VDISC veri seti bölünmüş kayıt sayıları

| | Fonksiyon Sayısı |
|---|---|
| Eğitim (Train) | 1 019 471 ( %80) |
| Validation (Doğrulama) | 127 476 ( %10) |
| Test | 127 419 ( %10) |

Draper VDISC veri seti içeriği incelendiğinde barındırdığı beş sınıfa ait veri sayılarının ve true/false oranlarının oldukça dengesiz olduğu görülmektedir. Öyle ki CWE-469 kodlu hata sınıfı için test verisetinde 127419 fonksiyon içinden sadece 278 tanesi hatalı olarak tanımlanmıştır ve bu hata kodu barındıran fonksiyonlar diğer hata kodları ile de yüksek oranda çakışmaktadır. Veri setindeki dengesizliğin giderilerek modelin başarımının daha doğru değerlendirilmesi amacıyla her bir sınıfa ait kayıt sayısının eşitlenmesi için iki farklı yöntem uygulanmıştır. Bu amaçla ele alınan veri setindeki veri miktarında azaltma yapılarak her bir hata sınıfındaki veri sayıları eşitlenmeye çalışılmıştır. Veri setinde azaltma yapıldığında veri sayısında büyük oranlarda azalma olmuştur. Bu durum veri setindeki bazı hata sınıflarındaki *True* sayılarının çok düşük olmasından kaynaklanmaktadır. Veri setindeki dengesizlik giderilse de büyük miktarda verinin kaybedilmesi ve az miktardaki veri ile çalışmanın yapılması sağlıklı görülmemiştir. Bu sebeple veri setine dengeli bir yapı oluşturmak amacıyla kayıt azaltma ve kayıt çoğaltma yöntemlerinin her ikisi ile de elde edilen veri setleri üzerinde deneyler yapılmıştır.

Azaltma işlemi orijinal veri seti üzerinde bulunan verilerin içerisinde en az sayıda hata barındıran sınıftakine eşit ya da yakın olacak şekilde seçilmesi şeklinde uygulanmıştır. Böylece her bir sınıfta yaklaşık aynı sayıda hatalı veri olan dengeli bir veri seti elde edilmiştir. Bu kapsamda hiç bir hata içermeyen satırlar kapsam dışında kalmıştır.

Çoğaltma işlemi orijinal veri setindeki hata sınıflarında yer alan veri sayısı diğerlerine göre az olan sınıflardakilerin çoğaltılması şeklinde yapılmıştır. Çoğaltma işlemi ile veri setindeki hata sınıfları birbirine yakın sayılara ulaşmıştır. Veri setindeki sayıları eşitlerken her bir hata sınıfında veri tekrarlaması yapılmıştır.

Bu kapsamda Draper VDISC verisetinden elde edilen üç farklı veri seti şu şekildedir;

1. Draper VDISC orijinal veri seti
2. Kayıtlar azaltılarak hata sayıları yaklaşık olarak eşitlenen veri seti
3. Kayıtlar çoğaltılarak hata sayıları yaklaşık olarak eşitlenen veri seti

### 4.1.2. GHPR veri seti

GHPR veri seti, GitHub'da yer alan toplamda 3026 hata düzeltmesi barındıran 307 Java projesinden elde edilmiştir. Bu veriseti Çizelge 4.4. ile verildiği gibi kaynak kod, düzeltmeye neden olan değişiklik ve proje bilgileri de dahil olmak üzere 16 özelliği içermektedir. Veri seti 3026 hata içeren, 3026 hata içermeyen olmak üzere toplam 6052 örnek veriden oluşmaktadır. Yazılımsal hatanın birden fazla dosyayı ilgilendiriyor olması göz önünde bulundurularak elde edilen bu örneklerin tek dosyada yapılan değişiklikleri barındırması gerektiği düşünülerek bu şekilde filtrelenmiştir. Oluşturulan verisetine GitHub[1] üzerinden erişilebilmektedir (Xu vd., 2020).

### 4.1.3. CoDesc veri seti

Kaynak kod ile ilgili olarak sektörel ve akademik anlamda artan ilgiye rağmen derin öğrenme gibi çok büyük veriye dayalı öğrenme modelleri için yeterli büyüklükte bir kaynak kod veri setinin bulunmaması sonucunda CoDesc veri seti oluşturulmuştur. Bu veri seti 4,2 milyon Java metodu içermektedir (Hasan vd., 2021). Alsuhaibani vd., 2018; Mnih ve Kavukcuoglu, 2013; Pennington vd., 2014 gibi çalışmalarda görülebileceği gibi, korpus boyutu büyüdükçe orantılı olarak metin gömme performansları da artmaktadır. Buradan yola çıkarak, bu boyuttaki verilerle eğitilen vektörleştirici modeller, kaynak kodlarının vektörel temsillerini daha uygun bir şekilde temsil edebilir. Aynı zamanda genel bir model oluşturularak yeni yapılan geliştirmeler için daha kapsayıcı bir yol izlenerek

[1]https://github.com/feiwww/GHPR_dataset

Çizelge 4.4. GHPR veri seti içerik tanımları

| Özellik | Açıklama |
|---|---|
| PROJECT_NAME | Projenin adı |
| PROJECT_OWNER | Projenin sahibi |
| PROJECT_DESCRIPTION | Projenin açıklaması |
| PROJECT_LABEL | Proje sahibi tarafından eklenen etiketler |
| PROJECT_LANGUAGE | Projenin geliştirildiği programlama dili |
| SHA_FIXED | Hata düzeltildikten sonraki sürüme ait ID |
| SHA_BUG | Hata düzeltilmeden önceki sürüme ait ID |
| DIFF_CODE | Hata ile ilişkili kod değişikliği |
| COMMIT_DESCRIPTION | Hataya ilişkin PR'a ait commit mesajı |
| COMMIT_TIME | Hataya ilişkin PR'a ait commit mesajının zamanı |
| OLD_CONTENT | Hata düzeltilmeden önceki kod dosyasının içeriği |
| NEW_CONTENT | Hata düzeltildikten sonraki kod dosyasının içeriği |
| OLD_PATH | Değişen dosyasının eski konumu |
| NEW_PATH | Değişen dosyasının yeni konumu |
| PR_TITLE | PR başlığı |
| PR_DESCRIPTION | PR açıklaması |

sıklıkla karşılaşılabilecek yeni kod eklenmesi durumunda ince ayar ve yeniden eğitim maliyetlerinin ortadan kaldırılması amaçlanmaktadır.

### 4.1.4. CommentFinder

CommentFinder veri seti GitHub'da yer alan 4901 ve Gerrit'de yer alan 6388 proje olmak üzere toplam 11289 projeye ait 151019 kaynak kod-inceleme çifti içermektedir. Geliştiricinin kendisi tarafından yazılan, botlar tarafından yazılan, ingilizce olmayan ve alakasız görülen yorumlar (örn; "it looks good to me", "thank you" vs.) veri setine dahil edilmemiştir. Tüm bu koşullar altında elde edilen veri seti incelendiğinde kayıtların %7,5 kadarının birbirleriyle aynı gözden geçirme yorumuna sahip olduğu görülmektedir (Hong vd., 2022). Bu veri seti ölçümleri yaparken kullanılan metriklere göre geçmiş çalışmalar arasında en iyi kod gözden geçirme performansını gerçekleştiren çalışma tarafından kullanılmıştır.

### 4.1.5. OGU-CENG-SC veri seti

İlk olarak, çalışmada kullanılmak üzere GitHub kod deposunda yer alan açık kaynak projelerde açılmış kod değişiklik istekleri ve ilişkili yorumlar belirlenen bazı kriterlere istinaden toplanmıştır. Bu kriterler;

- Açık ve erişilebilir olmak
- Java projesi olmak
- 50'den fazla yıldız almış olmak

şeklinde belirlenmiştir. Bu şekilde filtrelendikten sonra toplanan projelere[2] ait çekme isteklerinde yer alan kod değişiklikleri ve bir kod satırına doğrudan yapılmış yorumlar toplanarak veri seti oluşturulmuştur.

İkinci olarak, Bölüm 4.1.2. ile verilen GHPR veri seti tarafından dahil edilmiş projeler dikkate alınarak kod değişiklik istekleri ve ilişkili yorumlar toplanmıştır. GHPR veri setinde kod değişiklikleri yer almasına rağmen bu değişikliklere bağlı yapılan geliştirici yorumları olmadığı için yeniden veri toplama gereksinimi doğmuştur. Bu şekilde GHPR veri setinin genişletilmiş bir sürümünün oluşturulduğu söylenebilir. Bu şekilde toplanan veri setinde 470338 geliştirici yorumu görülmektedir. GHPR verisetinden farklı olarak tek bir dosya değişikliği gibi bir kısıtlmaya gerek duyulmadığından ve yorumlar doğrudan kod satırlarına yapıldığından sayıca çok daha büyük bir veri seti olan OGU-CENG-SC Veri Seti ortaya çıkmıştır.

Verilerin toplanması için kullanılan veri toplama aracı python dilinde geliştirilmiştir ve GitHub üzerinde yayınlanmıştır[3]. GitHub kod deposundan kaynak kod değişikliğine ilişkin veriler detaylı olarak alınabilmektedir. Bu çalışma kapsamında kaynak kod değişikliği ve yapılan yorum ikilisi yeterli olmakla birlikte gerekli durumlarda manuel kontroller yapabilmek adına ek bilgiler de toplanmıştır. Toplanan kayıtlara ait üst veri bilgileri ve açıklamaları Çizelge 4.5. ile verilmiştir.

[2]https://github.com/search?q=is%3Apublic+stars%3A%3E50&type=repositories&s=stars
[3]https://github.com/ykartal/Github-SourceCode-Review/blob/master/1-gather-dataset.py

Çizelge 4.5. Oluşturulan veri seti içerik tanımları

| Özellik | Açıklama |
|---|---|
| url | Yapılan yoruma ait bağlantı adresi |
| diff_hunk | Yapılan kod değişikliğine ait işaretlenmiş kaynak kod içeriği |
| path | Değişikliğe uğrayan .java dosyasının projedeki konumu |
| body | Kod değişikliğine istinaden yapılan geliştirici yorumu |
| pull_request_url | Kod değişikliğinin yer aldığı çekme isteğinin bağlantı adresi |
| start_line | Java dosyasındaki değişikliğin başlangıç satır numarası |

Örnek veri kaydına Şekil 4.2. ile yer verilmiştir. JSON olarak elde edilen verilere ait diff_hunk ve body alanları bu çalışmaların yapılması için yeterli bilgiyi sunmaktadır.


```
"url":"https://api.github.com/repos/firebase/firebase-android-sdk/pulls/comments/408345119",
"diff_hunk":"@@ -15,13 +15,21 @@\n package com.google.firebase.installations;\n \n import android.text
"path":"firebase-installations/src/main/java/com/google/firebase/installations/Utils.java",
"body":"a) please no empty line around imports\r\nb) should imports be ordered alphabetically? if yes,
"pull_request_url":"https://api.github.com/repos/firebase/firebase-android-sdk/pulls/1462",
"start_line":23
```


Şekil 4.2. Örnek veri kaydı

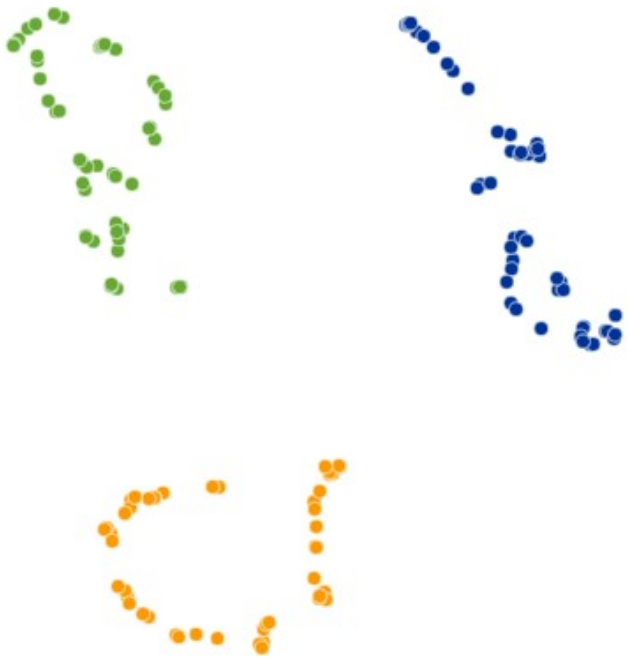
## 4.2. T-SNE (t-Distributed Stochastic Neighbor Embedding)

T-SNE (t-Distributed Stochastic Neighbor Embedding), çok boyutlu verilerin görselleştirilmesi için kullanılan bir makine öğrenimi yöntemidir. T-SNE, yüksek boyutlu verilerin içerdiği karmaşık yapıları daha düşük boyutlu bir uzayda temsil etmeyi hedeflemektedir (Van der Maaten ve Hinton, 2008).

T-SNE, veri noktalarını genellikle 2 veya 3 boyutlu olmak üzere daha düşük boyutlu bir uzaya yerleştirmektedir, böylece bu uzayda benzerlikler ve ilişkiler görsel olarak daha iyi anlaşılabilir hale gelmektedir. Özellikle görselleştirmelerde kullanılan T-SNE, veri noktalarının birbirine olan benzerliklerini ve gruplanmalarını korumaya çalışmaktadır (Van der Maaten ve Hinton, 2008).

T-SNE, veri noktalarının benzerliklerini ölçmek için istatistiksel olarak analiz edilebilen ancak tam olarak tahmin edilemeyen bir yaklaşım kullanmaktadır. İlk olarak, yüksek boyutlu uzayda her veri noktası birbirine olan benzerliğini hesaplar ve bu benzerliklerden oluşan bir olasılık dağılımı oluşturur. Daha sonra, düşük boyutlu uzayda noktaların birbirlerine olan benzerliklerini ölçer ve bu benzerlikleri de olasılık dağılımı olarak hesaplar. T-SNE, yüksek boyutlu uzaydaki benzerlikleri düşük boyutlu uzaydaki benzerliklere dönüştürmek için Kullback-Leibler (KL) divergence adı verilen bir istatistiksel ölçüm kullanmaktadır (Van der Maaten ve Hinton, 2008).

T-SNE'nin avantajlarından biri, yüksek boyutlu verilerdeki karmaşık yapıları ve kümeleme desenlerini görsel olarak ortaya çıkarabilmesidir. Bu sayede, veri setindeki anlamlı gruplanmaları veya benzerlikleri kolaylıkla gözlemleyebilme imkanı sunmaktadır. Ancak, T-SNE'nin dezavantajı, hesaplama maliyetinin yüksek olması ve bazı durumlarda veri setinin özelliklerine bağlı olarak farklı sonuçlar elde edilebilmesidir. Basit bir sınıflandırma problemi için Wattenberg vd., 2016 tarafından sunulan örnek bir görselleştirme Şekil 4.3. ile verilmektedir.



Şekil 4.3. T-SNE görselleştirme örneği (iterasyon: 5000, perplexity: 5, learning rate: 10)

## 4.3. Verilerin Ön İşlemden Geçirilmesi

Veri setleri her ne kadar çeşitli kriterler doğrultusunda oluşturulsa da veri setlerinde yer alan bazı kayıtların dışlanması gereken durumlar olabilmektedir. Bu çalışma içerisinde kaynak kod ve yorum çiftlerine bakıldığında bazı yorumların bir ya da birkaç kelimeden oluşan bir düzeltme önerisi içermeyen tebrik ya da övgü mesajı gibi standart yorumlar olduğu görülmüştür. Bu şekilde bulunan kayıtların dikkate alınmaması için düzenlemeler yapılmıştır. Ayrıca kaynak kod tarafında boş kod satırları ya da tek harf içeren kod satırları da aynı şekilde çalışma kapsamına alınmamıştır.

Bununla birlikte benzer kaynak kodların daha doğru saptanması için metinsel veriler içerisinde yer alan kaynak kodlar üzerinde bazı düzenlemelerin yapılması gerekebilmektedir. Kaynak kod içerisinde yer alan değişken, fonksiyon ya da terim adı gibi geliştiriciye bağlı olan ve teknik olarak önem arz etmeyen terimlerin dışlanması ya da uygun şekilde değiştirilmesi çeşitli çalışmalarda bir ön işlem olarak uygulanmış yöntemler arasındadır. Bu tez kapsamında uygulanan ön işlemlerin önerilerin doğruluğunu etkilediği görülmüştür.

### 4.3.1. Sözcüksel analiz (Lexical analysis)

Sözcüksel analiz bir programlama dilinde yazılmış kaynak kodlar üzerinde gerçekleştirilen bir işlemdir. Sözcüksel analiz ile kaynak kod parçalanarak anahtar kelimeler, operatörler ve değişkenler gibi belirteçlere ayrılmaktadır. Bu belirteçler programlama dilinin yapı taşlarıdır. Her programlama dilinin kendi karakteristik özellikleri olması sebebiyle sözcüksel analiz açısından programlama dilleri arasında farklı yaklaşımlar olması gerekmektedir. Örneğin python programlama dilinde her bir sekme ya da boşluk özel bir anlam içerdiğinden bu boşlukların ya da sekmelerin sözcüksel analiz sırasında bir şekilde işaretlenmesi gereklidir (Vikipedi, 2022). Sözcüksel analizin birincil hedefleri şu şekilde sıralanabilir (Bergmann, 2003);

1. **Belirteç Tanıma:** Sözcüksel analiz, kaynak kodda yer alan her bir belirteci saptamaya yardımcı olur. Belirteçler "if", "for", "while" gibi anahtar sözcükleri, değişken adları gibi tanımlayıcıları, "+" veya "-" gibi operatörleri ve sayılar ya da dizeler gibi sabit değerleri içerebilir. Derleyici bu belirteçleri tanımlayarak, kodun yapısı hakkında bir yaklaşım ortaya konmasını sağlar.

2. **Hata Algılama:** Sözcüksel analiz, kaynak koddaki sözcüksel hataların veya sözdizimi hatalarının saptanmasında hayati bir rol oynar. Bir belirteç tanınmazsa veya karakterlerin düzenlenmesi programlama dilinin sözcük kurallarını ihlal ederse, bu durumu saptayabilir. Örneğin, bir programcı bir anahtar kelimeyi yanlış yazarsa veya geçersiz bir karakter kullanırsa, sözcüksel analiz bu hatalı durumu işaretleyebilir.

3. **Sembol Çizelgesi Oluşturma:** Sözcüksel analiz için zorunlu olmasa da bir sembol çizelgesi, derleyiciler ve yorumlayıcılar tarafından kaynak koddaki değişkenler ya da fonksiyonlar gibi çeşitli semboller hakkında bilgi depolamak için kullanılan bir veri yapısıdır. Sözcüksel analiz, bir kaynak koddaki belirteçleri tanımlayarak ve bu belirteçleri veri türleri vb. öznitelikleri ile ilişkilendirerek sembol çizelgesinin oluşturulmasına ve doldurulmasına yardımcı olur. Simge çizelgesi tanımlayıcının adı, türü, kapsamı ve bellek konumu gibi verileri barındırabilir.

4. **Kod optimizasyonu:** Sözcüksel analiz, fazla veya gereksiz belirteçleri belirleyerek kod optimizasyonuna da katkıda bulunabilir ayırca gereksiz boşlukları ya da karakterleri kaynak koddan temizleyebilir. Örneğin, kullanılmayan değişkenleri veya başvurulmayan işlevleri tanımlayabilir ve programcıya olası optimizasyon alanları hakkında bilgi verebilir.

Genel olarak sözcüksel analizin amacı, kaynak kodun yapılandırılmış bir temsilini sağlamaktır böylece hataları belirlemek ve derleme sürecinin sonraki aşamaları için gerekli bilgiler sağlanmış olunur. Örnek kodlar üzerinde sözcüksel analiz ile oluşturulmuş çıktılar Çizelge 4.6. ile verilmiştir.

Kod benzerliklerine göre tespitte bulunmak için tasarlanan bir sistemde bir ön işlem olarak değerlendirildiğinde sözcüksel analizden geçirilen bir kaynak kod içeriğinde değişken isimleri gibi tanımlayıcıların temizlenmesi benzerlik oranlarının arttırılmasını sağlayabilir. Değişken isimlerinin benzerlik üzerindeki etkisi azaltılırken vektörize işlemleri sırasında göz ardı edilmesi olası olan ancak önem arz edebilecek operatörlerin de etiketlenerek öne çıkması sağlanabilir. Bu şekilde uygulanan bir iyileştirme benzer kaynak kodların eşleştirilmesinde pozitif bir rol oynayabilir. Tez sürecinde sözcüksel analizin etkisini araştırmak için GitHub'da yer alan Sözcüksel Analizci[4] kullanılmıştır.

[4]https://github.com/merveceyhan/Lexical-Analysis

Çizelge 4.6. Kaynak kod ve karşılık gelen sözcüksel analiz çıktıları

| **Kaynak Kod** | **Sözcüksel Analiz Çıktısı** |
|---|---|
| public class OnSelectHandler | (public, Id) (class, CI) (OnSelectHandler, Id) (In, Symb) |
| Crashlytics. logException(throwable); | (Crashlytics, Id) (., Symb) (logException, Id) (( Symb) (throwable, Id) 0, Symb) (;, Symb) (In, Symb) |
| private static List String> reverse View(List<String> input) ( | (private, Id) (static, Id) (List, Id) (<, Symb) (String, Id) (>, Symb) (reverse View, Id) (( Symb) (List, Id) (<, Symb) (String, Id) (>, Symb) (input, CI) 0, Symb) (& Symb) (In, Symb) |
| jdbcTemplate.getConnection0 .prepareCall( "EXECsp_configure' clrenabled',0; RECONFIGURE;"); | (jdbcTemplate, Id) (., Symb) (getConnection, Id (( Symb) 0, Symb) (., Symb) (prepareCall, Id) (( Symb) |

### 4.3.2. Kaynak kod soyutlama (Source code abstraction)

Kaynak kodun ham hali üzerinde çalışarak kod desenlerini öğrenmek oldukça zor bir süreçtir. Programlama dili tarafından tanımlanmış anahtar kelimeler, operatörler vs. yanında bağımsız kullanılan büyük bir kelime dağarcığı bulunması ve kullanılması sebebiyle zorlu bir problem olarak karşımıza çıkmaktadır. Bu sebeple Tufana vd. tarafından, verilen java kaynak kodunun soyutlanarak kelime sınırlı ama aynı ifadeyi taşıyan bir temsilinin oluşturulması için src2abs[5] adını verdikleri açık kaynak koduna sahip bir araç geliştirilmiştir. Bu araç parametre aldığı java kaynak kodunda parçalama işlemleri yaparak kullanıcı tanımlı değişken ismi, metot ismi gibi değerleri METHOD_1, VAR_1 gibi sayısal olarak artarak giden değerler ile değiştirir. Bu değerleri orijinal değerleri saklayarak sağlar. Böylece örneğin aynı metot ile karşılaşırsa yine METHOD_1 değerini kullanabilir. Bu yaklaşım aynı zamanda soyutlanmış bir kaynak kodun orijinal haline geri döndürülebilmesine imkan sağlamaktadır. Bazı özel durumlar incelenerek bu durumlar ile karşılaşıldığında kaynak kodun soyutlanması engellenmiştir. Örneğin "size", "add" gibi sıkça kullanılan ve kod yazma sürecinde anlamlı ifadeler barındıran isimlendirmeler olduğu gibi bırakılmıştır (M. Tufano, Watson, Bavota, Penta, vd., 2019).

[5]https://github.com/micheletufano/src2abs

Orijinal java kaynak koduna karşılık abs2src ile soyutlanmış kaynak kod ikilisi örneği şu şekilde verilmiştir;

```
//Orijinal Java Kaynak Kodu
public PageProperties getProperties(){
    if(hasProperties()){
        return properties;
    } else {
        return null;
    }
}
```

```
//Soyut Java Kaynak Kodu
public TYPE_1 METHOD_1( ) { if(METHOD_2 ( ) ) {
return properties ; } else { return null; } }
```

## 4.4. Veri sayısallaştırma (vectorization) ve metin gömme (text embeddings)

Kullanılacak derin öğrenme, makine öğrenmesi ya da benzerlik algoritmalarının kaynak kodu işleme alabilmesi için ilgili kaynak kodların sayısal gösterimlerine ihtiyaç duyulmaktadır. Bu sebeple bu metinsel verilerin sayısallaştırılması yani vektörlere dönüştürülmesi bir gereklilik olarak karşımıza çıkmaktadır. Vektörlere dönüştürülmesi aşamasında uygulanan yöntemin hem zaman maliyeti hem de doğruluğa etkisi önem arz etmektedir.

Veri sayısallaştırma, verileri matematiksel vektörler olarak temsil etme işlemidir. Başta makine öğrenimi ve veri analizi alanlarında sıkça kullanılan bir teknik olmakla birlikte metin sınıflandırma, görüntü tanıma, öneri sistemleri gibi geniş bir alanda kullanılmaktadır. Görüntüler, sesler, çizelgeler gibi farklı türde veriler makine öğrenmesi gibi yaklaşımlar tarafından işlenmeye başlanmadan önce sayısallaştırılarak vektörler haline getirilerek kullanılır. Metin veriler de diğer türlerdeki veriler gibi vektörlere dönüştürüldükten sonra işlenmeye alınmaktadır (Singh ve Shashi, 2019).

Bu tez kapsamında metin verilerin işlenmesi sözkonusu olduğundan metin verilerin vektörlere dönüştürülmesi konusu ön plana çıkmaktadır. Metin vektörleştirme, metinlerin içerdikleri kelimeler veya kelime öbeklerinin sayısal değerlere yani vektörlere dönüştürülmesi ile sağlanır. Bu vektörler, metin verinin yapısını ya da özelliklerini yansıtır. Yaygın olarak kullanılan metin vektörleştirme yöntemleri Kelime Torbası (Bag-Of-Words(BoW)), TF-IDF, Kelime Gömme (Word

Embeddings), N-Gram Vektörler, Karakter Gömme (Char-Level Embeddings), Belge Gömme (Document Embeddings) şeklinde sıralanabilir (Aubaid ve Mishra, 2020).

### 4.4.1. Kelime torbası (Bag of words (BoW))

Kelime torbası, metin işleme ve doğal dil işleme alanlarında yaygın olarak kullanılan bir yöntemdir. Metin belgelerini temsil etmek için kullanılan bir modeldir. Bu modelde, bir metin belgesi, içerdiği kelimelerin frekansına dayalı olarak bir vektörle temsil edilir. Kelime sırası veya dilbilgisi kuralları göz ardı edilir, sadece kelime frekansları dikkate alınır. Örneğin, "Bu bir örnek cümle" ifadesi, "bu", "bir", "örnek", "cümle" kelimelerinden oluşan bir vektörle temsil edilir.

Kelime torbası oluşturmak için öncelikle, temsil etmek istenilen metin belgesi alınır. Bu belge, bir yazı, makale, e-posta veya başka bir metin içeriği olabilir. Elde edilen belgeyi kelimelerine ayırmak için bir parçalama işlemi uygulanır. Bu sırada belgede yer alan metin boşluklardan ve noktalama işaretlerinden arındırılmış olunur. Bu şekilde toplanan tüm benzersiz kelimeler ile bir sözlük oluşturulur. Her bir benzersiz kelime için benzersiz ve indeksi ifade eden bir kimlik oluşturulur ve eşlenir. Sözlükte yer alan her benzersiz kelime için kelimenin belgede kaç kez tekrarlandığını ifade eden frekans hesaplanır. Böylece belgenin kelime frekans vektörü oluşturulur. Tüm metin belgelerini temsil eden kelime frekans vektörleri birleştirilerek bir "Kelime Torbası" oluşturulur. Böylece her bir metin belgesinin her benzersiz kelimenin frekansını gösterdiği bir matris elde edilmiş olunur (Qader vd., 2019).

Kelime Torbası, basit, ölçeklenebilir ve hesaplama verimliliği gibi avantajlar sunmaktadır. Tüm bu özellikleri ile metin sınıflandırma, duygu analizi, bilgi çıkarma gibi bir çok alanda yaygın olarak kullanılmaktadır (Qader vd., 2019).

### 4.4.2. Terim frekansı - ters doküman frekansı (TF-IDF)

Terim Frekansı - Ters Doküman Frekansı (TF-IDF), doğal dil işleme ve bilgi erişim alanlarında yaygın olarak kullanılan istatistiksel bir yöntemdir. Bu yöntem ile bir dokümanda geçen bir kelimenin önemi belirlenmeye çalışılır. TF-IDF, bir terimin bir dokümandaki görülme sıklığı (TF) ile tüm dokümanlardaki görülme sıkılığını (IDF) birleştirerek bu terimin genel önemini hesaplar. Yalnızca tek bir dokümana odaklanmak yerine tüm dokümanlardaki terimleri de göz önünde bulundurması sebebiyle önemli sonuçlar sağlamaktadır (Havrlant ve Kreinovich, 2017).

**Terim Frekansı(TF)**: Bir dokümandaki bir terimin ne kadar sık geçtiğini gösteren ölçüdür. Dokümanın içeriğindeki her bir terimin frekansı hesaplanır, bu değer bu dokümana özgü bir değerdir. TF, dokümandaki bir kelimenin tekrarlanma sayısının toplam kelime sayısına bölümü ile hesaplanır. $d$ dokümanı için $t$ terimin tekrarlanma sayısı, $w$ toplam kelime sayısı olmak üzere TF değerini hesaplayan formül 4.1 ile verilmiştir (Havrlant ve Kreinovich, 2017).

$$TF(t,d) = \frac{t}{w} \tag{4.1}$$

**Ters Doküman Frekansı(IDF)**: Bir terimin tüm dokümanlardaki görülme sıklığını gösteren ölçüdür. Bir terimin ne kadar nadir veya popüler olduğunu belirlemek için kullanılır. Hesaplanan IDF değeri ne kadar yüksekse ilgili terime o derece nadir rastlanmaktadır. Yani terim dokümanların çoğunda bulunuluyorsa IDF değeri düşer. IDF değeri, toplam doküman sayısının terimi içeren doküman sayısına bölümünün logaritması alınarak hesaplanır. Yani $N$ toplam doküman sayısını, $df$ ilgili terimi içeren doküman sayısını ifade etmek üzere IDF değeri formul 4.2 ile hesaplanır (Havrlant ve Kreinovich, 2017).

$$IDF(t) = log\frac{N}{1+df} \tag{4.2}$$

**TF-IDF Hesaplama**: Bir terimin *TF-IDF* değeri, terimin *TF* değeri ile *IDF* değeri çarpılarak elde edilir. Bu, terimin dokümandaki sıklığını belirleyen *TF* ile genel önemini belirleyen *IDF* arasında bir denge sağlar. Eğer bir terim bir dokümanda sıkça geçiyorsa ve tüm dokümanlarda de yaygın ise, *TF-IDF* değeri düşük olacaktır. Öte yandan, bir terim ne kadar az dokümanda tekrarlanıyorsa *TF-IDF* değeri o kadar yüksek olacaktır. Bu sayede, dokümanların genelinde önemli terimlerin vurgulanması sağlanır. TF-IDF değerini hesaplayan formül 4.3 ile verilmiştir (Havrlant ve Kreinovich, 2017).

$$TF - IDF = TF(t,d) * IDF(t) \tag{4.3}$$

### 4.4.3. Word2Vec

*Word2Vec*, doğal dil işleme alanında kullanılan ve kelime vektörlerinin dağıtımsal temsilini öğrenmek için kullanılan popüler bir tekniktir. Yöntem, büyük

metin veri setlerinden öğrenilen vektörler aracılığıyla kelimelerin anlamsal ilişkilerini yakalamayı hedefler. *Word2Vec*, özellikle kelime benzerliği, kelime ilişkileri ve anlamsal olarak benzerlik ölçümleri gibi görevler için etkili bir yöntem sağlamaktadır. *Word2Vec*, kelime vektörleri oluştururken "benzer kelimeler benzer bağlamlarda kullanılır" fikrine dayanan dağıtımsal bir yaklaşımı benimser. Bu sayede, benzer anlamlara sahip kelimelerin vektörleri birbirlerine yakın konumda olurken, semantik olarak farklı kelimelerin vektörleri arasında anlamsal mesafeler korunur. *Word2Vec*, bu özellikleri ile kelime düzeyindeki doğal dil işleme görevlerinde, metin sınıflandırmadan kelime önerme sistemlerine kadar birçok alanda kullanılmaktadır (Turner vd., 2017).

### 4.4.4. Doc2Vec

*Doc2Vec*, doğal dil işleme alanında ve belgelerin dağıtımsal temsillerini öğrenmek için kullanılan bir yöntemdir. *Word2Vec'*in genişletilmiş bir versiyonu olarak kabul edilir. *Doc2Vec*, bir belgenin içeriğini tanımlamak için vektörler kullanmakta ve belgeler arasındaki anlamsal ilişkileri yakalamayı amaçlamaktadır (H. Zhang ve Zhou, 2019).

*Doc2Vec*, kelime düzeyindeki temsillerin yanı sıra belge düzeyindeki temsilleri de öğrenir. Bu, belgelerin birbiriyle benzerliklerini ve ilişkilerini yakalamak için oldukça kullanışlı bir yöntemdir. Örneğin, *Doc2Vec* uygulanan bir veri setinde benzer konulara sahip belgelerin vektörlerinin birbirlerine yakın konumda olmaları beklenmektedir.

*Doc2Vec*, genellikle büyük veri setleri ya da daha doğru bir deyişle büyük metin koleksiyonları üzerinde eğitilir ve öğrenilen belge vektörleri, metin sınıflandırma, belge gruplama, metin benzerliği ve öneri sistemleri gibi birçok doğal dil işleme görevinde kullanılır. Bu yöntem, belgelerin anlamsal yapısını ve ilişkilerini tespit etme konusunda etkili bir araç sağlamaktadır ve makine öğrenimi tabanlı doğal dil işleme çalışmaları için faydalı bir bileşen olarak kullanılmaktadır (H. Zhang ve Zhou, 2019).

### 4.4.5. Transformer mimarisi

*Transformer mimarisi*, doğal dil işleme (Natural Language Processing (NLP)) ve diğer dizi tabanlı görevler için kullanılan bir derin öğrenme modeli ve mimarisidir. *Transformer mimarisi*, özellikle dil modelleri ve tercüme modelleri oluşturma gibi görevlerde büyük bir başarı elde etmiştir. Bu, önceki yöntemlerin

sınırlamalarını aşan ve daha iyi sonuçlar sağlayan yenilikçi bir yaklaşım olarak görülmektedir. Son zamanlarda oldukça popüler olan yapay zeka uygulaması ChatGPT[6] de Transformer'ın dil işleme yeteneklerini kullanarak, gerçek zamanlı sohbetler için önceden eğitilmiş bir model olarak karşımıza çıkmaktadır.

*Transformers* mimarisi, dil anlamını yakalamak için dikkat mekanizmasını (Attention Mechanism) kullanmaktadır. Dikkat mekanizması, bir kelimenin veya bir dizi öğenin diğer öğelerle olan bağlantısını hesaplama yeteneğine sahip sinir ağı mekanizmasıdır. Bu, bir kelimenin anlamını belirlemek için kelimenin bağlamını ve diğer kelimelerle olan ilişkisini dikkate alabilen bir yapı sağlamaktadır (Vaswani vd., 2017).

Transformers, birçok önemli bileşenden oluşmaktadır. İlk olarak, bir dizi "encoder" katmanı bulunmaktadır. Her *encoder* katmanı, giriş verilerini dikkat mekanizmasıyla işler ve önceki katmanın çıktılarına dayalı olarak yeni bir tanım oluşturur. Bu, kelimelerin veya öğelerin bağlamsal ilişkilerini anlamak için bilgi birikimi sağlar (Vaswani vd., 2017).

Ayrıca, *Transformer*'da "decoder" katmanları da bulunmaktadır. *Decoder*, genellikle çeviri veya ardışık tahmin gibi özgün bir çıktı üretimi yapan görevlerde kullanılmaktadır. *Decoder* katmanları, kodlama katmanlarının çıktılarına dayanarak sonuçları oluşturur. Dikkat mekanizması, özellikle decoder katmanları arasında önemli bir rol üstlenmektedir çünkü bu katmanlar sırayla çıktılar üretirken, dikkat mekanizması önceki çıktıları dikkate alarak doğru bağlamın yakalanmasını sağlamaktadır (Vaswani vd., 2017).

*Transformer* mimarisi, dil modelleri için çok sayıda parametreye sahip olabilir ve büyük ölçekli eğitim veri setleri gerektirebilir. Bununla birlikte, önceden eğitilen *Transformer* modelleri, çeşitli görevlerde bağlam ve kurallara bağlı kalmak üzere optimize edilerek bu görevlere uyarlanabilir. Bu, dil anlama, metin sınıflandırma, metin üretimi, metin eşleştirme ve diğer doğal dil işleme görevlerinde etkili bir şekilde kullanılabilmesini sağlar.

*Transformer* mimarisi, 2017 yılında "Attention Is All You Need" başlıklı bir makale ile duyurulmuş ve o zamandan beri doğal dil işleme alanında devrim niteliğinde bir etkiye sebep olmuştur (Vaswani vd., 2017). *Transformers*, özellikle BERT, GPT ve XLNet gibi önceden eğitilmiş modellerle birlikte, birçok dil işleme

[6]https://openai.com/chatgpt

görevinde oldukça iyi sonuçlar elde edilmesini sağlamıştır. Bu sebeple araştırma ve endüstride büyük ilgi görmüştür.

#### 4.4.5.1. Dikkat mekanizması (Attention mechanism)

*Dikkat mekanizması*, doğal dil işleme ve makine çevirisi gibi alanlarda kullanılan bir derin öğrenme mekanizmasıdır. Bu yöntem, bir girdi dizisindeki farklı öğelerin birbirleriyle olan ilişkisini hesaplamak ve modelin dikkatini önemli öğelere odaklamak için kullanılmaktadır.

*Dikkat mekanizması*, bir öğenin diğer öğelerle olan ilişkisini ölçmek için bir ağırlık vektörü kullanmaktadır. Bu ağırlık vektörü, dikkat ağırlıkları olarak adlandırılmakta ve her bir öğe arasındaki önemi belirtmektedir. Bunu sağlamak için her bir öğeyle ilgili bir ağırlık vektörü hesaplanmaktadır ve daha sonra bu ağırlık vektörleri toplanarak öğelerin birleşik temsilini oluşturmaktadır.

*Dikkat mekanizması* genellikle *sorgu (query)*, *anahtar (key)* ve *değer (value)* vektörleri şeklinde 3 temel bileşenden oluşmaktadır. *Sorgu* vektörü, *dikkat mekanizmasının* odaklanacağı öğeyi temsil ederken, *anahtar* vektörleri diğer öğeleri temsil etmekte ve değer vektörleri, dikkat ağırlıklarıyla çarpıldıktan sonra birleşik temsili oluşturmaktadır.

*Dikkat mekanizması*, önceki öğelerin bilgisini kullanarak her bir öğenin dikkat ağırlığını hesaplar. Dikkat ağırlıkları, çeşitli benzerlik ölçüleri kullanılarak hesaplanabilmektedir. Örneğin, yaygın olarak kullanılan bir dikkat mekanizması tipi olan *dot-product attention*, sorgu vektörü ve anahtar vektörleri arasındaki iç çarpımı kullanarak dikkat ağırlıklarını hesaplamaktadır (Luong vd., 2015).

*Dikkat mekanizması*'nın, özellikle uzun ve bağımlılık ilişkileri içeren dizilerde (örneğin, uzun cümlelerde veya metin belgelerinde) etkili bir şekilde çalıştığı görülmüştür. Bu yaklaşım, modelin belirli bir öğeye odaklanmasını ve öğeler arasındaki bağlantıları dikkate almasını sağladığı için önemli öğeleri tanıyabilmekte ve çıktıyı daha iyi şekillendirebilmektedir.

*Dikkat mekanizması*, derin öğrenme modellerinde yaygın olarak kullanılan bir teknik olarak karşımıza çıkmaktadır. Özellikle dil modelleri, çeviri modelleri, metin sınıflandırma ve metin üretimi gibi doğal dil işleme görevlerinde önemli bir rol oynamaktadır. Ayrıca, dikkat mekanizması, önceki bölümde değinildiği gibi

*Transformer* mimarisi gibi birçok ileri derin öğrenme modelinin temel bir parçasıdır ve bu modellerin başarısını büyük ölçüde artırmıştır.

#### 4.4.5.2. BERT (Bidirectional encoder representations from transformers)

*BERT*, doğal dil işleme alanında çığır açan bir derin öğrenme modelidir. *BERT*, doğal dili kelime ve cümle düzeyinde anlamak için eğitilen bir modeldir. Bu model, hem sol hem de sağ bağlamları dikkate alarak kelimeleri işleme alır. Bunu yaparken önceki ve sonraki kelimelerin bağlamsal ilişkilerini anlamak için biçimsel bir dil tahmin göreviyle eğitilir.

*BERT*, *Transformer* adı verilen *Dikkat Mekanizması*'na dayanan bir yapı kullanmaktadır. Önceki bölümde detayları verildiği üzere *Dikkat Mekanizması*, kelime ya da cümle içindeki farklı kısımların birbirleriyle etkileşimini modellemek için kullanılan bir sinir ağı mekanizmasıdır. Bu sayede *BERT*, uzun bağımlılıkları ve belirli bir kelimenin bağlamını daha iyi anlayabilmektedir.

*BERT*, Öneğitim ve İnce Ayarlama (Fine-Tuning) olmak üzere genellikle iki aşamada eğitilir. Öneğitim aşamasında, büyük miktarda metin verisi kullanarak dil modeli eğitilir. Ardından, eğitilmiş model, çeşitli doğal dil işleme görevleri için İnce Ayarlama aşamasında kullanılabilir. İnce ayarlama aşamasında, model, görevin gerektirdiği etiketleme yapısı ve metin verisi kullanılarak görev özgü veri seti üzerinde eğitilir. Bu sayede, model, belirli bir görevi çözebilecek şekilde özelleştirilir (Devlin vd., 2018).

*BERT*, birçok doğal dil işleme görevinde etkileyici sonuçlar elde etmiştir. Kelime seviyesinde kelime tahmini, metin sınıflandırma, metin eşleştirme, duygu analizi, soru-cevap sistemleri ve dil tabanlı görevler gibi birçok görevde başarılı performans sergilemektedir. Ayrıca, tek bir modelin çoklu dilleri desteklemesi ve genel doğal dil anlamını yakalama yeteneği, *BERT*'i çok yönlü ve geniş kapsamlı bir dil modeli haline getirmiştir.

*BERT*'in başarısının temelinde, önceki dil modellerinden farklı olarak, biçimsel bir dil tahmin görevi için çift yönlü (bidirectional) bir yaklaşım kullanması bulunmaktadır. Bu, bir kelimenin anlamını belirlemek için hem önceki hem de sonraki kelimelerin bağlamını kullanabilmesi anlamına gelir. Bu çift yönlü modelleme, kelimelerin bağlamsal ilişkilerini daha iyi anlamasını sağlar.

BERT, dil anlamını yakalama yeteneği ve çoklu dilleri destekleme özelliği ile ön plana çıkar. Eğitim verisi için çok büyük veri kümeleri kullanıldığından, BERT genellikle büyük hesaplama kaynakları gerektirmektedir. Ancak, bir kez eğitildikten sonra, bu model, çeşitli doğal dil işleme görevlerinde etkili bir şekilde kullanılabilmektedir.

Bu sonuçlar doğrultusunda *Bert*, birçok uygulama ve araştırma çalışmasının temelini oluşturmuştur. Özellikle, daha önce zorlu kabul edilen dil anlama görevlerinde büyük ilerlemeler sağlanmış ve birçok gelişmiş doğal dil işleme modelinin temelini oluşturmuştur.

#### 4.4.5.3. CodeBert

*CodeBERT*, kod tabanlı programlama görevleri için özel olarak tasarlanmış bir dil modelidir ve doğal dil işleme yöntemlerini kod tabanlı senaryolara uyarlar. *CodeBERT*, aslında önceki bölümde detayları verilen *BERT* modelinin kod tabanlı versiyonudur ve bir prgoramlama diline ait kaynak kodu anlamak için eğitilmiştir. Yazılım geliştirme için kullanılan programlama dilleri ve bu dillere özgü kaynak kod yapısının özelliklerini dikkate alır. *CodeBERT*, genellikle kaynak kod metinlerini anlamak, kaynak kod parçalarını sınıflandırmak, kod önerileri sağlamak ve kod benzerliğini ölçmek gibi görevlerde kullanılmaktadır (Z. Feng vd., 2020).

*CodeBERT*, kaynak kodların anlamsal yapısını çıkarmak için derin öğrenme tabanlı bir dil modeli kullanmaktadır. Bu sayede, kaynak kod tabanlı metinlerdeki anlamı ve bağlamsal ilişkileri anlama yeteneğine sahip olur. *CodeBERT*, önceden eğitim ve ince ayarlama aşamalarını kullanarak belirli bir programlama dili ya da belirli bir kaynak kod tabanlı görev için özelleştirilebilmektedir.

*CodeBERT*, yazılım mühendisliği alanında bir dizi görevde kullanılmıştır (Pan vd., 2021). Örneğin, kod önerileri sağlamak, kod parçalarını sınıflandırmak, hata tespiti yapmak ve kod benzerliği analizi gibi görevlerde etkili olabilme yeteneğine sahiptir. *CodeBERT*, yazılım geliştirme süreçlerinde verimliliği artırmak ve kod kalitesini iyileştirmek için kullanılan bir araç olarak değerlendirilebilir.

## 4.5. Benzerlik Modeli

### 4.5.1. Vektör farkları

*Vektör Farkları*, iki ya da daha fazla vektör arasındaki benzerlik veya farklılık derecesini ölçmek için kullanılan bir kavramdır. *Vektör Farkları*, özellikle veri madenciliği, makine öğrenimi ve örüntü tanıma gibi alanlarda sıklıkla kullanılan bir tekniktir.

*Vektör Farkları*, vektörlerin uzayda geometrik konumları arasındaki uzaklıkları temsil eder. Basitçe; vektörler, n-boyutlu uzayda noktalar olarak düşünülebilir ve vektör farkları, bu noktalar arasındaki uzaklığı ölçer. *Vektör Farkları*, genel olarak vektörlerin büyüklüğüne ve yönelimine bağlı olarak hesaplanmaktadır.

Metin benzerliği hesaplamalarındaki temel mantık, metnin vektörler olarak temsil edilmesidir ve böylece bu vektörler arasındaki mesafelerin farklı yöntemlerle ölçülmesine dayanmaktadır.

En yaygın kullanılan vektör mesafesi ölçüleri şunlardır:

**Öklid Mesafesi**: Öklid mesafesi, iki vektör arasındaki doğrudan öklid uzaklığını temsil eder. İki vektör arasındaki Öklid mesafesi, her boyutun farkının karelerinin toplamının karekökü olarak hesaplanır (Kabak vd., 2017). Örneğin; p ve q noktaları arasındaki öklid uzaklığı (4.4) eşitliği ile hesaplanabilir.

$$\Delta d = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{4.4}$$

**Manhattan Mesafesi**: Manhattan mesafesi için İki nokta arasındaki uzaklık, bu noktalardan geçen ve dik kesişen doğru parçalarının uzaklığı kadardır. İki vektör arasındaki Manhattan mesafesi, her boyutun mutlak farklarının toplamı olarak hesaplanır (Kabak vd., 2017). Örneğin; p ve q noktaları arasındaki Manhattan uzaklığı (4.5) eşitliği ile hesaplanabilir.

$$\Delta d = \sum_{i=1}^{n} |p_i - q_i| \tag{4.5}$$

**Kosinüs Benzerliği**: Kosinüs benzerliği, iki vektör arasındaki açının kosinüsüne dayanır ve vektörler arasındaki yönelim benzerliğini ölçer. İki vektör arasındaki kosinüs benzerliği, iki vektörün iç çarpımının vektörlerin normlarının çarpımına bölünmesiyle hesaplanır (Kabak vd., 2017). Örneğin; p ve q noktaları arasındaki Kosinüs uzaklığı (4.6) eşitliği ile hesaplanabilir.

$$\Delta d = \frac{\sum_{i=1}^{n}(p_i q_i)}{\sqrt{\sum_{i=1}^{n}(p_i)^2}\sqrt{\sum_{i=1}^{n}(q_i)^2}} \tag{4.6}$$

Vektör farkları, veri analizinde birçok farklı uygulama için kullanışlı çözümler sunmaktadır. Örneğin, sınıflandırma problemleri ve benzerlik tabanlı arama işlemleri gibi birçok alanda vektör farkları kullanılarak benzerlik veya farklılık ölçüleri elde edilebilmektedir.

Sonuç olarak; *Vektör Farkları*, vektörlerin benzerlik veya farklılık derecesini hesaplamak için kullanılan çeşitli tekniklerin genel adıdır. Çeşitli veri setleri üzerinde farklı benzerlik hesaplamaları daha doğru sonuçlar elde edilmesine sebep olabilmektedir.

Tez çalışması, kaynak kod metni üzerinde çeşitli aşamalarda çeşitli yöntemler uygulayarak kod benzerliklerini ölçmeyi amaçlamaktadır. Üzerinde çalışılan test setindeki her kaynak kodun vektörel gösterimi ve kod inceleme geçmişindeki her kaynak kodun vektörel gösterimi arasındaki vektör farklarının hesaplanması için Kosinüs, Öklid ve Manhattan yöntemleri kullanılmış ve en başarılı sonucun elde edilmesi için vektöre çevirme ve vektör farkı hesaplama işlemleri birlikte tüm kombinasyonları ile deneylerde kullanılmıştır. Bu kombinasyonlardan elde edilen sonuçlar karşılaştırılmıştır. En başarılı yöntem için, tüm test seti için metin değerlendirme metriklerinin ortalaması ve tam eşleşmelere ait sonuçlar dikkate alınmıştır.

### 4.5.2. Annoy (Approximate nearest neighbors (Yaklaşık en yakın komşular))

*Annoy*, büyük boyutlu veri kümesi üzerinde hızlı ve etkili bir şekilde benzerlik tabanlı arama yapmayı sağlayan bir indeksleme ve arama kütüphanesidir. Annoy, yüksek boyutlu veri setlerinde yakın komşuları hızlı bir şekilde bulmak için kullanılmaktadır (W. Li vd., 2019).

Annoy Index[7], büyük boyutlu vektörlerin benzerlik tabanlı arama için kullanıldığı durumlarda özellikle etkilidir. Örneğin, görüntü işleme, doğal dil işleme veya kullanıcı profilleri gibi uygulamalarda, büyük boyutlu vektörlerle çalışmak yaygındır ve bu tür verilerde benzerlik tabanlı arama yapmak zorlu bir görev olabilir. Annoy Index, bu tür durumlarda hızlı ve etkili bir çözüm sunar.

Annoy Index'in çalışma prensibi, verilerin yüksek boyutlu vektörler olarak temsil edilmesine dayanır. İlk olarak, veri kümesi önceden belirlenmiş bir boyutta bir vektör uzayına yerleştirilir. Ardından, bu vektörler arasındaki benzerlikleri hesaplamak için bir ölçü kullanılır. Örneğin, Kosinüs Benzerliği veya Öklid Mesafesi gibi ölçüler sıklıkla kullanılan yöntemlerdendir.

Annoy Index, veri kümesini bir ağaç yapısı şeklinde organize eder. Her düğüm, vektörlerin bir alt kümesini temsil eder ve alt düğümler de benzerliklerine göre sıralanır. Bu ağaç yapısı, yakın komşuları hızlı bir şekilde bulmayı sağlar. Arama işlemi sırasında, sorgulanan vektöre en yakın komşuları hızlı bir şekilde tespit etmek için ağaç yapısı kullanılır.

Annoy Index'in avantajlarından biri, yaklaşık en yakın komşuları bulmak için kullanılan hızlı bir hesaplama yöntemi olan hesaplama yakınsama algoritmasını (Approximation Search Algorithm) kullanmasıdır. Bu yöntem, tam olarak en yakın komşuları bulmak yerine, yakınsama ile sonuçları hızlı bir şekilde yaklaşık olarak hesaplar. Bu, büyük boyutlu veri kümelerinde hızlı arama işlemlerini mümkün kılar.

Annoy Index, açık kaynak bir kütüphanedir ve birçok programlama dilinde kullanılabilir. Örneğin, Python, C++, Java, ve Go gibi dillerde desteklenir. Kullanımı kolaydır ve büyük boyutlu veri kümesinde benzerlik tabanlı aramalar yapmak için birçok seçenek ve yapılandırma seçeneği sunar.

Sonuç olarak, Annoy Index, büyük boyutlu vektörler üzerinde hızlı ve etkili benzerlik tabanlı arama yapmayı sağlayan bir indeksleme ve arama kütüphanesidir. Büyük boyutlu veri kümesiyle çalışan uygulamalarda benzerlik tabanlı arama ihtiyacını karşılamak için kullanılır ve hesaplama yakınsama algoritmasını kullanarak hızlı arama işlemleri sağlar.

[7]https://github.com/spotify/annoy

## 4.6. Sonuç Önerme

Ham verinin vektörlere dönüştürülmesi ve vektör farkları ile en benzer kayıtların bulunması her ne kadar hedef verinin saptandığı algısı yaratsa da vektör mesafelerinin hiç bir şekilde kabul edilemeyecek farklar ile saptanmış olması yani aslında aranan kaynak kod için hiç bir önerinin olmaması olasılığı bulunmaktadır. Bu sebeple benzerlik oranı için belirlenecek bir eşik değerin altında yer alan değerlere sahip bir verinin en benzer kayıt olarak önerilmesi uygun görünmemektedir. Bu sebeple vektör farkı için daha doğrusu benzerlik değeri için bir eşik değerinin belirlenmesi ve bu eşik değerinin altındaki benzerliğe sahip kayıtlar için bir öneri yapılmaması gerekir.

Vektör farkları ile ilgili teknikler için bir eşik değerinin belirlenmesi konusunda kesin bir yöntem bulunmamaktadır. Genel yaklaşım belirlenecek eşik değerinin çalışılan konu ve veriye göre değişiklik gösterebileceği şeklindedir. Bu sebeple farklı çalışmalarda farklı hesaplamalar ile eşik değerinin belirlenmesi gerektiği düşünülebilir.

Bu tez kapsamında eşik değerinin belirlenmesi için veri setinde yapılan deneyler sonucunda elde edilen veriler referans alınarak normal dağılım hesaplamaları ve skorların türevinin alınması yapılmıştır. Standart sapma ve güven aralıkları göz önünde bulundurularak empirik kural adı verilen yöntem ile 1-sigma değerine karşılık gelen verinin %68'i kapsanacak şekilde bir eşik değeri belirlenmiştir.

### 4.6.1. Normal dağılım

Normal dağılım, istatistik ve olasılık teorisi alanlarında sıkça kullanılan bir olasılık dağılımıdır (Vikipedi, 2023). Normal dağılımın temel özellikleri şunlardır:

1. **Simetri**: Normal dağılım, orta noktasında simetriktir. Ortalama değer, dağılımın simetri eksenidir ve dağılımın sağ ve sol tarafı aynıdır.
2. **Merkezi Limit Teoremi**: Normal dağılım, merkezi limit teoremi ile ilişkilidir. Merkezi limit teoremi, bağımsız ve aynı dağılıma sahip rassal değişkenlerin toplamının, n büyüdükçe normal dağılıma yaklaştığını belirtir. Bu nedenle, birçok doğal olay ve süreç, normal dağılıma yakın bir şekilde dağılabilir.
3. **Belli bir ortalama ve standart sapma**: Normal dağılımın şekli, ortalama değer ve standart sapma parametreleri tarafından belirlenir. Ortalama,

dağılımın merkezini belirlerken, standart sapma, dağılımın yayılma derecesini ifade eder. Daha düşük standart sapma, daha dar ve yüksek bir dağılımı temsil ederken, daha yüksek standart sapma, daha geniş ve düşük bir dağılımı temsil eder.

4. **Çan şeklinde dağılım**: Normal dağılımın grafiksel temsili, bir çan şeklinde olup iki yana doğru simetridir. Dağılımın zirvesi orta noktadadır ve sağa ve sola doğru yayılan kuyrukları vardır. Dağılımın çan şekli, rastgele olayların çoğunun orta nokta etrafında gerçekleştiğini ve ekstrem olayların daha az olası olduğunu gösterir.

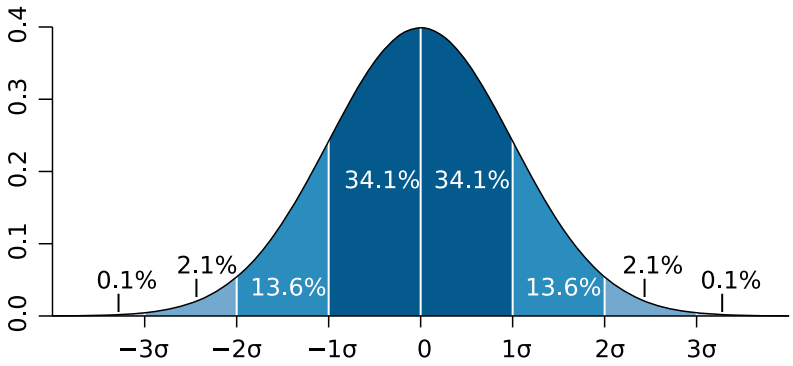
**Standart sapma**, bir veri setinin yayılma veya dağılım ölçüsüdür. Bir veri setindeki değerlerin ortalama değerden ne kadar uzaklaştığını gösterir. Standart sapma, verilerin ne kadar değişken olduğunu ve dağılımın yayılma derecesini ölçer. Daha yüksek bir standart sapma, verilerin daha geniş bir aralığa yayıldığını gösterirken, daha düşük bir standart sapma, verilerin daha yakın bir aralıkta odaklandığını gösterir. Standart sapma, veri setinin istatistiksel analizinde ve karar verme süreçlerinde yaygın olarak kullanılır (Vikipedi, 2023).

**Güven aralığı**, ise bir örneklem veya popülasyon parametresi (genellikle ortalama) hakkında bir tahmin yaparken belirsizlik düzeyini ifade eder. Bir örneklem üzerinden hesaplanan bir parametre değeri, gerçek popülasyon parametresine çok yakın olabilir, ancak tam olarak aynı olmayabilir. Güven aralığı, tahminin belirli bir güven düzeyiyle ne kadar doğru olabileceğini ifade eder.

Güven aralığı, bir alt sınırla bir üst sınırdan oluşur ve genellikle yüzde olarak ifade edilir. Örneğin, %90 güven düzeyinde bir güven aralığı, tahminin gerçek parametre değerini içerme olasılığının %90 olduğunu gösterir. Güven aralığı genellikle örneklem büyüklüğü, standart sapma ve güven düzeyine bağlı olarak hesaplanır.

Güven aralığı, istatistiksel analizlerde ve karar verme süreçlerinde kullanılır. Bir parametrenin güven aralığının dar olması, tahminin daha kesin olduğunu gösterirken, geniş bir güven aralığı, tahminin daha az kesin olduğunu gösterir. Genellikle, daha yüksek bir güven düzeyi kullanılarak hesaplanan güven aralığı, daha geniş olacaktır çünkü daha yüksek güven düzeyi daha fazla belirsizlikle ilişkilidir.

Bir normal dağılımdan seçilmiş değerlerin %68'i ortalamanın bir standart sapma uzaklığındaki noktalar arasındadır; değerlerin neredeyse %95'i ortalamadan iki standart sapma uzaklıklar aralığında; ve %99,7'si üç standart sapma uzaklıklar aralığında bulunur. Buna *empirik kural* adı verilir. Bu durumu Şekil 4.4. ile gösterilmektedir (Vikipedi, 2023).




Şekil 4.4. Standart sapma ve güven aralıkları

Bu tez kapsamında yapılan çalışmada 1-sigma ($1\sigma$) güven aralığı referans alınarak veri seti üzerinde yapılan çalışmalar sonucu ortaya çıkan benzerlik oranlarından en yüksek ilk %68'i öneri olarak sunulmuştur. Belirlenen %68 kayıt arasında kalan en düşük değer eşik değeri olarak kabul edilebilir.

### 4.6.2. Türev

Türev, bir fonksiyonun değişim hızını temsil eden bir kavramdır. Bir fonksiyonun türevinden faydalanılarak pik noktaları bulunabilmektedir. Fonksiyonun yönünün ve eğiminin analiz edilmesi için sıklıkla kullanılan bir yöntemdir. Türevin pozitiften negatife geçtiği noktalar fonksiyonun değerinin maksimum değer aldığı noktalardır. Değişim hızını temsil eden türevi kullanarak eşik değerin belirlenmesi kullanılabilecek bir yöntemdir.

Bu tez kapsamında yapılan çalışmada benzerlik oranlarına karşılık gelen Bleu skorlarının grafiği üzerinde türev alındığında ortaya çıkan türevin pozitiften negatife döndüğü ve benzerlik için 0.65 değerini aldığı nokta eşik değeri olarak kullanılabilir.

## 4.7. Deneysel Çalışmalarda Kullanılan Ölçüm Yöntemleri

Tasarlanan sistemin başarısı, doğru sınıfa atanan örnek sayısı ve yanlış sınıfa atanan örnek sayısı nicelikleri ile ölçülür. Testler sonucunda elde edilen sonuçların başarım bilgileri hata matrisi ile belirtilmektedir. Çizelge 4.7. ile örnek olarak verilen hata matrisinde yer alan sütunlar test kümesindeki örneklere ait gerçek sayıları ifade etmektedir. Satırlar ise modelin tahminlemesini göstermektedir.

Çizelge 4.7. İki sınıflı hata matrisi örneği

| | | Gerçek Sınıflar | |
|---|---|---|---|
| | | Sınıf 1 | Sınıf 2 |
| Tahmini Sınıflar | Sınıf 1 | 7 | 1 |
| | Sınıf 2 | 3 | 8 |

Sınıflandırma ölçümlerinde sıklıkla kullanılan ve Çizelge 4.8. ile verilen metriklerden doğru/yanlış ve pozitif/negatif terimlerini şu şekilde açıklayabiliriz;

**TP (True-Positive)**: Modelin doğru olarak önerdiği bir verinin gerçekte de doğru olduğu durumu ifade eder.

**FP (False-Positive)**: Modelin doğru olarak önerdiği bir verinin gerçekte yanlış olduğu durumu ifade eder.

**FN (False-Negative)**: Modelin yanlış olarak önerdiği bir verinin gerçekte doğru olduğu durumu ifade eder.

**TN (True-Negative)**: Modelin yanlış olarak önerdiği bir verinin gerçekte de yanlış olduğu durumu ifade eder.

Başarıyı ölçmek için kullanılan en yaygın ve en temel yöntem, modelin doğruluğudur. *Doğruluk (ACC)*, doğru kategorilere ayrılmış olan örnek sayısının (TP+TN), toplam örnek sayısına olan (TP+TN+FN+FP) oranıdır(4.7). *Hata oranı* ise bu değerin 1'e tamlayanıdır. Diğer bir ifadeyle yanlış sınıflandırılmış örnek sayısının (FP+FN), toplam örnek sayısına (TP+TN+FN+FP) oranıdır.

Çizelge 4.8. Doğru/Yanlış ve Pozitif/Negatif ölçümlerinde kullanılan hata matrisi gösterimi

| | | Gerçek Sınıflar | |
|---|---|---|---|
| | | Doğru | Yanlış |
| Tahmini Sınıflar | Doğru | TP (True Positive) | FP (False Positive) |
| | Yanlış | FN (False Negative) | TN (True Negative) |

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.7}$$

*Kesinlik (PPV)*, bir sınıf için doğru tahminlenmiş örnek sayısının (TP), ilgili sınıf için tahmin edilmiş tüm örnek sayısına (TP+FP) oranıdır (4.8).

$$PPV = \frac{TP}{TP + FP} \tag{4.8}$$

*Duyarlılık (TPR)* ise doğru sınıflandırılmış pozitif örnek sayısının (TP), toplam pozitif örnek sayısına (TP+FN) oranıdır (4.9).

$$TPR = \frac{TP}{TP + FN} \tag{4.9}$$

*F1 Puanı* kesinlik ve duyarlılığın harmonik ortalamasıdır ve 4.10 eşitliğinde verildiği şekilde hesaplanabilir.

$$F_1 = \frac{2TP}{2TP + FP + FN} \tag{4.10}$$

### 4.7.1. Tam eşleşme (Exact match/Perfect prediction)

Test verisine ait bir kayıt için önerilen metin ile beklenen metin birebir eşleşiyorsa buna tam eşleşme yani mükemmel tahmin adı verilmektedir.

Bu tez kapsamında otomatik kaynak kod gözden geçirme deneyleri ölçümlerinde kullanılan kriterlerden biri de mükemmel tahminler yani tam eşleşmelerdir. Bu tahminler, sistemin gözden geçirme yorumu olarak önerdiği ve test verisinde yer alan kaynak kod için gerçekte yapılan kod gözden geçirme

yorumlarının tam olarak yani harfi harfine aynı olduğu tahminlerdir. Özellikle tekrarlayan ve yorum alan kaynak kodların gözden geçirme yorumları otomatikleştirildiğinde tam eşleşmelerin görülmesi beklenir. Farklı yazılım ekiplerine ait farklı standartlarda kod inceleme verileri üzerinde deneyler yapılsa da, mükemmel tahminlerin varlığı, farklı ekipler tarafından yazılan farklı projelerdeki farklı standartlara rağmen benzer hataların yapılabileceğini göstermektedir.

### 4.7.2. İşlem süresi (Execution time)

*İşlem süresi*, girdi olarak alınan verinin çeşitli adımlardan geçirilip işlenmesi ve nihayetinde önerinin üretilmesi için geçen zamandır.

Hong vd., 2022 tarafından yapılan çalışmada; verimliliğin, Bilgi erişim(Information Retrieval(IR)) tabanlı kod inceleme sistemleri ile kod inceleme yorumları oluşturan diğer sistemler arasındaki önemli farklardan biri olduğunu ortaya koymaktadır. Yaptığımız deneylerde farklı modellerin test verileri ile tahmin üretirken harcanan işlem süreleri hesaplanmıştır. Kod gözden geçirme geçmişindeki tüm kodlar, tüm vektörleştirici yöntemleriyle işleme alınmıştır. Her modelin uygulanma biçimleri farklı olduğundan işlem sürelerinin hesaplanma yöntemleri de farklılık gösterebilmektedir. Metin verilerin vektörlere dönüştürülmesi dikkate değer bir zaman maliyeti oluşturmamakla birlikte vektörlerin hangi yöntem ile oluşturulduğu modelin öneri üretme zamanını etkileyebilmektedir. Bunlarla birlikte sistemin en çok zaman harcadığı kısım vektör farklarının ölçüldüğü adımdır. İşlem süresi adıyla tanımladığımız bu metrik, tüm işlemin bu bölümünde harcanan zamanı temsil etmektedir.

İşlem süresi sistemin üzerinde çalıştığı donanım ile de doğrudan ilgili olmakla birlikte bu tez kapsamında kullanılan sistem bilgileri şu şekildedir;

- Intel Core i7-10750H; 16GB Ram; NVIDIA GTX1650; Windows 11
- GPU Sunucu Bilgisayarı (SuperServer 4028GR-TRT/ NVIDIA Quadro P5000 GDDR5x)

### 4.7.3. Doğal dil işleme problemlerinde kullanılan ölçüm yöntemleri

Özellikle sınıflandırma problemleri için eşleşmeler doğru/yanlış şeklinde ikili değerler ile elde edilmektedir. Böylece rutin bir değerlendirme sürecinde önerilen çıktıların beklenen çıktılar ile aynı olması ya da olmaması beklenir. Ancak doğal dil işleme problemlerinde önerilen metinlerin birebir eşleşmesi her zaman beklenmez. Bazı durumlarda metin verilerin kelime benzerliği olarak ya da özellikle anlamsal olarak benzer ifadeler ile önerildiği durumlar oldukça fazladır. Bu sebeple metin çıktılar üreten doğal dil işleme problemlerinde test verilerinden elde edilen öneriler değerlendirilirken bu konuda özel olarak geliştirilmiş yöntemler kullanılmaktadır.

Metin verilerinin karşılaştırılmasında kullanılan yaygın ölçüm yöntemleri arasından BLEU, METEOR ve ROUGE bu tez kapsamında da kullanılan metriklerdir. Geçmiş çalışmalar ile karşılaştırmaların daha sağlıklı yapılabilmesi adına önceki çalışmalarda sıklıkla kullanılan BLEU skoru özellikle önemli görülmektedir ve bu tez kapsamında ana değerlendirme ölçütü olarak belirlenmiştir. Kod gözden geçirme yorumlarının yer aldığı test veri setindeki kodlara en çok benzeyen k adet kaynak kod seçildikten sonra bu kodlara yapılan yorumların beklenen yoruma yakınlığı bu metriklerle ölçülür ve böylece kullanılan yöntemin başarısını ölçmek amaçlanmıştır. Önceki çalışmalarda, kod gözden geçirme yorumu oluşturma ve IR gibi yöntemlerinin sonuçları bu metrikler yardımıyla değerlendirilmiştir. Nihayetinde istenen ölçüm, otomatik kod gözden geçirme yorumu önerisinin manuel olarak yapılan yoruma ne kadar yakın olabileceğidir. Tüm kombinasyonların kullanıldığı deneylerde, tüm test verileri için her bir metriğin ortalaması ve tam olarak eşleşen yorumların sayısı ayrı ayrı hesaplanmıştır. Birden fazla öneri olması durumunda metrik puanı en yüksek olan önerinin puanı o grubun puanı olarak alınmıştır.

#### 4.7.3.1. BLEU (Bilingual evaluation understudy)

*BLEU*, özellikle metin çevirisi alanında yaygın olarak kullanılan bir metrik olarak kabul edilmektedir. BLEU skoru, metin çevirisi için bir sistem tarafından üretilen çıktının, bir veya daha fazla referans çeviriye ne kadar benzediğini ölçer. Yani temel olarak bir sistem çıktısı ile referans çeviriler arasındaki benzerliği hesaplamaktadır (Papineni vd., 2002). Benzerlik düzeyini bir skor ile verir. İki ana bileşeni vardır;

1. **N-gram Yakınlığı**: BLEU, n-gram yakınlığı üzerine kuruludur. *N-gramlar*, metindeki ardışık n kelime dizisini temsil eder. Örneğin, 2-gram, metindeki ardışık 2 kelimeyi ifade eder. Sistem çıktısındaki n-gramlar ile referans çevirideki n-gramlar arasındaki eşleşmelerin sayısı hesaplanır.
2. **Kesinlik (Precision)**: Kesinlik, sistem çıktısında bulunan n-gramların referans çevirideki n-gramlarla olan eşleşmelerine dayanarak hesaplanır. Bu, sistem çıktısının ne kadar doğru olduğunu gösterir.

BLEU skoru, n-gram kesinlik değerlerinin geometrik ortalamasını alarak hesaplanır. Ayrıca, çeviriye olan vektör uzaklığını ve özetleme problemlerinde n-gramları temsil etmek için modifikasyonlar da yapılabilir (Papineni vd., 2002).

BLEU skoru, 0 ile 1 arasında bir değer alır. 1, sistem çıktısının referans çeviriyle tam olarak örtüştüğünü gösterirken, 0 hiçbir örtüşme olmadığını belirtmektedir. Yüksek BLEU skorları, daha iyi bir çeviri önerildiğini göstermektedir (Papineni vd., 2002).

BLEU, metin çevirisi alanında yaygın olarak kullanılan bir ölçüm yöntemidir, ancak bazı sınırlamaları vardır. Örneğin, yalnızca kelime düzeyinde benzerlikleri hesaplar ve dil yapısı, anlam ilişkileri ve anlam bütünlüğü gibi daha yüksek seviyeli dil özelliklerini dikkate almaz. Ayrıca, referans çevirilerin sayısı ve kalitesi de sonuçları etkileyebilir.

BLEU tanımlanırken çeviri metinleri üzerinden örnekler verilse de metin karşılaştırma konusunda bir çok alanda kullanılan bir metriktir. Bu tez kapsamında deneylerde önerilen kod gözden geçirme yorumlarının test kaytlarındaki referans metin ile karşılaştırılması için kullanılmıştır. Başka çalışmalarda olduğu gibi test verisindeki tüm kayıtlar için önerilen kod gözden geçirme yorumlarına ait BLEU skorlarının ortalaması alınarak sonuçlar elde edilmiştir.

#### 4.7.3.2. METEOR (Metric for evaluation of translation with explicit ordering)

METEOR, metin çevirisi performansının değerlendirilmesi için kullanılan bir ölçüm yöntemidir (Banerjee ve Lavie, 2005). BLEU gibi, makine çevirisi alanında yaygın olarak kullanılan bir metriktir.

METEOR, sistemin ürettiği çıktı ile referans çeviriler arasındaki benzerliği değerlendirirken, bazı ek özellikleri de dikkate almaktadır. BLEU'dan farklı olarak,

sadece n-gram eşleşmelerine dayalı çıkarımlar yerine daha yüksek seviyeli dil özelliklerini de içermektedir. Yalnızca sözlüksel benzerlikleri değil, aynı zamanda üst düzey dilsel ve anlamsal özellikleri de dikkate alan kapsamlı bir değerlendirme sağlamayı amaçlamaktadır. Bu yaklaşım daha kaliteli bir değerlendirme sağlayabilmektedir (Banerjee ve Lavie, 2005). Meteor metriği, sistem çıktısı ile referans çeviriler arasındaki benzerliği ölçmek için çeşitli bileşenleri dikkate alır:

1. **Kesinlik**: METEOR, sistem çıktısı ve referans çeviriler arasındaki n-gram eşleşmelerini dikkate alarak kesinliği hesaplamaktadır. N-gramların örtüşmesini ölçerek sistem çıktısının ne kadar doğru olduğunu değerlendirmektedir.
2. **Unigram Recall**: METEOR, sistem çıktısı ve referans çeviriler arasındaki unigramların eşleşmelerini değerlendirerek recall değerini de değerlendirmeye dahil etmektedir. Bu, sistem çıktısında herhangi bir önemli kelimenin eksik olup olmadığının değerlendirilmesine yardımcı olur.
3. **İçerik ve Eşanlamlı Kelime Eşleştirme**: METEOR, kelime formları ve eşanlamlı kelimelerdeki varyasyonları işlemek için kök çıkarma algoritmaları ve eşanlamlı kelime eşleme teknikleri uygulayarak eşleştirme doğruluğunu artırmaya çalışmaktadır.

METEOR skoru, bu tekniklerin bir kombinasyonunu temsil eder ve 0 ile 1 arasında bir değer alır. Çevirinin sadece sözlüksel benzerliğini değil, akıcılığı, yeterliliği ve diğer dilbilimsel yönlerini de dikkate alarak daha kapsamlı bir değerlendirme sunmayı amaçlamaktadır. Yüksek Meteor skoru, daha iyi bir çeviri performansı anlamına gelmektedir.

#### 4.7.3.3. ROUGE (Recall-oriented understudy for gisting evaluation)

ROUGE metriği, doğal dil işleme ve metin özetleme görevlerinde yaygın olarak kullanılan bir değerlendirme ölçüsüdür. Önerilen özetlerin kalitesini bir veya daha fazla referans özetiyle karşılaştırarak değerlendirmeyi amaçlamaktadır (Lin, 2004).

ROUGE, sistem tarafından önerilen özet ile referans özeti arasındaki benzerliği, n-gramların örtüşmesine dayalı olarak ölçmektedir; burada n-gram, n tane kelimenin bitişik bir dizisidir (Lin, 2004). ROUGE'un üç ana çeşidi ROUGE-N, ROUGE-L ve ROUGE-S'dir.

1. **ROUGE-N**: Sistem özeti ile referans özeti arasındaki n-gram çakışmasını ölçen ROUGE tekniğidir. Çakışan n-gramların sayısına dayalı olarak kesinliği, duyarlılığı ve F1 puanını hesaplamaktadır (Lin, 2004).
2. **ROUGE-L**: Sistem tarafından önerilen özet ile referans özet arasındaki en uzun ortak alt diziyi ölçmektedir. Her iki özette de aynı sırada yer alan en uzun kelime dizisini dikkate almaktadır. Bu teknik, kelimelerin sırasını dikkate alarak cümle düzeyinde örtüşme ve benzerliğin nicel bir değerlendirmesini sağlamaktadır (Lin, 2004).
3. **ROUGE-S**: skip-bigram istatistiklerini, sistem tarafından yapılan öneriye ait skip-bigramlara ve referans özetlerine bakarak hesaplamaktadır. Sözcük düzeninde bazı varyasyonlara izin vererek anlamsal benzerliği yakalamaya çalışmaktadır (Lin, 2004).

ROUGE puanları da diğer metriklere benzer olarak 0 ile 1 arasında değişmektedir ve daha yüksek puanlar daha iyi özet kalitesini göstermektedir. Sistem tarafından önerilen özet ile referans özeti arasındaki örtüşme ve benzerliğin niceliksel bir ölçüsünü sunmaktadırlar. ROUGE, metin özetleme görevlerinde yaygın olarak kullanılmaktadır ve çeşitli araştırma yarışmalarında standart bir değerlendirme ölçütü olarak benimsenmiştir.

ROUGE'un bazı sınırlamaları olduğunu not etmek önemlidir. Yalnızca sözlüksel eşleştirmeye dayanır ve anlamsal veya bağlamsal anlayışı yakalamaz. Ek olarak, ROUGE puanları özet uzunluğu, referans kalitesi ve n-gram veya skip-bigram boyutu seçimi gibi faktörlere duyarlı olabilir. Bu nedenle, özetleme kalitesinin daha kapsamlı bir değerlendirmesini sağlamak için genellikle diğer ölçümler ve insan değerlendirmesiyle birlikte kullanılmaktadır.

Kelimelerin sırasını dikkate alarak cümle düzeyinde eşleşmelere odaklandığı için bu tez kapsamında ROUGE metrikleri arasından *ROUGE-L* metriği kullanılmıştır.

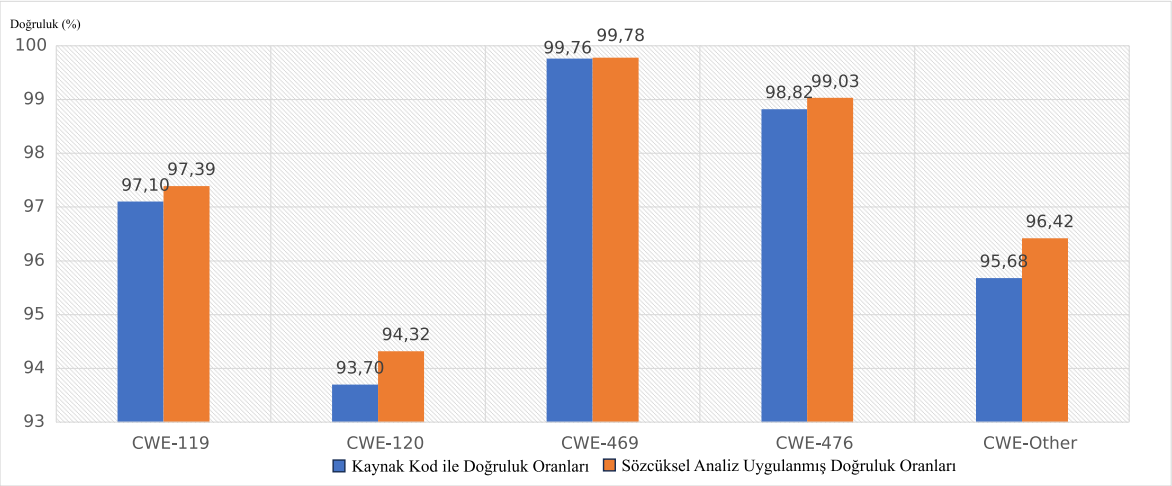
# 5. BULGULAR VE TARTIŞMA

Kaynak kodlar üzerinde yapılan bu tez çalışmasında temel amaç benzer kaynak kodların tespit edilmesi ve böylece daha önce hatalı ya da eksik olarak yorumlanan kaynak kod kesimlerinin yeni geliştirilen yazılımlar içerisinde ortaya çıkması durumunda erken teşhisini sağlamaktır. Bu sebeple geliştirilecek bir sistemin başarısı kaynak kodlar arasındaki benzerlik oranlarının iyi tespit edilmesiyle doğrudan ilişkilidir. Bu benzerliklerin yüksek doğruluk oranında tespit edilebilmesi için literatürde yer alan benzerlik algoritmalarının yanı sıra verilerin uygun şekilde temizlenmesi ve bu bölümde de değinilen bazı ön işlemlerin uygulanması uygun görünmektedir. Bu tez çalışmasında; farklı teknikler, farklı yaklaşımlar ve farklı yöntemler ile, etkilerinin ölçülmesi için deneylere eklenmiş ve elde edilen bulgulara bu bölümde yer verilmiştir.

## 5.1. Sözcüksel Analizin Etkileri

Sözcüksel analizin etkilerinin ölçülmesi için Draper VDISC veri seti kullanılmıştır. Bu veri seti üzerinde daha önce uygulanan çalışmadaki (Tanwar vd., 2020) derin öğrenme modeli referans alınarak her bir kayıt için doğrudan kaynak kodlar yerine sözcüksel analiz çıktılarının girdi olarak verilmesi sağlanmıştır. Elde edilen yeni metin değerler vektörlere dönüştürülerek derin öğrenme modelinin eğitilmesi amacıyla kullanılmıştır. Böylece vektörel çıktılar ve bu çıktılara karşılık gelen beş sınıflı bir çizelgenin derin öğrenmenin yapısına uygun olarak oluşturulması sağlanmıştır. Elde edilen sayısallaştırılmış vektör girdiler ile 6 adımdan oluşan evrişimsel sinir ağı modeli eğitilmiş ve test verisi üzerinde tahminler yapması sağlanmıştır. Sözcüksel analiz ile kaynak kod içerisinde yer alan ve en az seviyede teknik bilgi barındıran değişken isimleri, metot isimleri gibi değişkenlik gösterebilecek metinlerin tüm metin içerisindeki etkisi azaltılırken toplama, atama gibi çalışma zamanında yüksek etkiye sahip kelimelerin etkisinin arttırılması hedeflenmiştir.

Sözcüksel analizin uygulandığı ve uygulanmadığı durumların doğruluk değerlerini kıyaslayan grafik Şekil 5.1. ile verilmektedir. Buradan da görüldüğü üzere sözcüksel analizin uygulanması tüm sınıflarda doğruluk oranının artmasını sağlamıştır. Önceki çalışmalarda doğruluk oranı tüm sınıflarda %90'ın üzerinde olmasına rağmen sözcüksel analiz örneğin CWE-120 sınıfı için %93,70 olan değerin




Şekil 5.1. Sözcüksel analizin doğruluğa etkisi

%94,32'e yükselmesine sebep olmuştur. Benzer şekilde diğer sınıflar için de doğruluk oranında artış görülmektektedir.

Derin öğrenme modeli öncelikle üç farklı senaryo ile aynı veri seti üzerine sözcüksel analiz ön işlemine tabi tutulmadan doğrudan uygulanmıştır. İlk senaryo derin öğrenme modelinin beş farklı sınıfın her biri için ikili(true/false) çıktı üretmesi üzerine kuruludur. İkinci senaryo diğer sınıfların etkilerinin ortadan kaldırılması için tek bir sınıf(CWE-119) üzerinde ikili(true/false) çıktı üretmesi üzerine kuruludur. Üçüncü senaryo ise herhangi bir sınıfa ait hata olması durumunda tek bir ikili(true/false) çıktı üretmesi üzerine kurulmuştur. Bu sebeple üçüncü senaryo için ek bir sütunda herhangi bir sınıfta hata içeren satıra True değeri atanmış ve bu sütun referans alınarak çalışılmıştır. Bu senaryolar ile sonuçların elde edilmesinin ardından verisetine sözcüksel analiz ön işlemi uygulanarak aynı senaryolar tekrarlanmıştır. Birinci, ikinci ve üçüncü senaryolar sonucunda elde edilen doğruluk, kesinlik, duyarlılık ve F1 skoru değerleri ön işlem uygulanmış ve uygulanmamış olarak Çizelge 5.1.'te verilmiştir. Elde edilen çıktılarda doğruluk değeri yüksek çıkmasına karşılık kesinlik, duyarlılık ve F1 skor değerlerinin düşük kaldığı gözlenmiştir. Draper VDISC veri seti içeriği incelendiğinde barındırdığı beş sınıfa ait veri sayılarının ve true/false oranlarının oldukça dengesiz olduğu görülmektedir. Öyle ki veri setindeki dengesizlik durumu, CWE-469 hata kodu için hataların neredeyse %100 doğruluk ile saptanmış olunsa bile özellikle duyarlılık, kesinlik ve F1-skor değerlerinin sıfıra çok yakın hesaplanmasına sebep olmaktadır.

Çizelge 5.1. Sözcüksel analiz ön işleminin metrikler üzerine etkisi

| | | Senaryo 1 | | | | | Senaryo 2 | Senaryo 3 |
|---|---|---|---|---|---|---|---|---|
| | | CWE-119 | CWE-120 | CWE-469 | CWE-476 | CWE-other | CWE-119 | Hata Var/Yok |
| Orijinal Veri Seti | Doğruluk | 0,9620 | 0,9257 | 0,9976 | 0,9894 | 0,9554 | 0,9682 | 0,9071 |
| | Kesinlik | 0,2461 | 0,3038 | 0 | 0,3613 | 0,2315 | 0,3403 | 0,3717 |
| | Duyarlılık | 0,4714 | 0,7235 | 0 | 0,1694 | 0,2699 | 0,6937 | 0,6278 |
| | F1 Skoru | 0,3234 | 0,4279 | 0 | 0,2307 | 0,2492 | 0,4566 | 0,4670 |
| Sözcüksel Analiz Uygulanan Veri Seti | Doğruluk | 0,9739 | 0,9432 | 0,9978 | 0,9903 | 0,9642 | 0,9726 | 0,9160 |
| | Kesinlik | 0,3192 | 0,3116 | 0 | 0,3908 | 0,2973 | 0,3134 | 0,3531 |
| | Duyarlılık | 0,3115 | 0,3956 | 0 | 0,0645 | 0,2246 | 0,3523 | 0,3561 |
| | F1 Skoru | 0,3151 | 0,3486 | 0 | 0,1108 | 0,2559 | 0,3317 | 0,3546 |

Çizelge 5.2. Veri seti üzerinde yapılan dengeleme işleminin metrikler üzerine etkisi

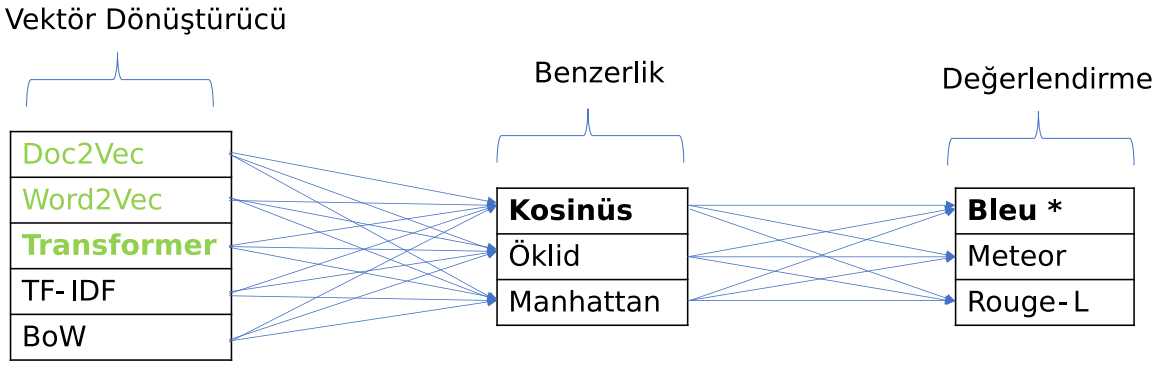
| | | CWE-119 | CWE-120 | CWE-469 | CWE-476 | CWE-other |
|---|---|---|---|---|---|---|
| Orijinal Veri Seti | Kesinlik | 0,32 | 0,31 | 0 | 0,39 | 0,30 |
| | Duyarlılık | 0,31 | 0,40 | 0 | 0,07 | 0,23 |
| | F1 Skoru | 0,32 | 0,35 | 0 | 0,11 | 0,26 |
| Arttırılmış Veri Seti | Kesinlik | 0,95 | 0,90 | 0,98 | 0,93 | 0,88 |
| | Duyarlılık | 0,49 | 0,65 | 0,67 | 0,91 | 0,60 |
| | F1 Skoru | 0,64 | 0,64 | 0,80 | 0,92 | 0,70 |
| Azaltılmış Veri Seti | Kesinlik | 0,10 | 0,77 | 0,70 | 0,70 | 0,65 |
| | Duyarlılık | 0,10 | 0,74 | 0,20 | 0,65 | 0,65 |
| | F1 Skoru | 0,10 | 0,74 | 0,25 | 0,66 | 0,66 |

Hem sınıf bazlı hem de hata bazlı dengesiz dağılımların modelin hata tespit oranlarını olumsuz etkilediği düşünülerek veri setindeki hata sınıflarına ait dengesizliklerin giderilmesi için verilerin arttırılması ve azaltılması ile iki yeni veri seti elde edilmiştir. Öyle ki CWE-469 kodlu hata sınıfı için test verisetinde 127 418 fonksiyon içinden sadece 278 tanesi (%0,2) hatalı olarak tanımlanmıştır ve bu hata kodu barındıran fonksiyonlar diğer hata kodları ile de yüksek oranda çakışmaktadır (Kartal vd., 2021). Bu durum CWE-469 sınıfı için kesinlik, duyarlılık ve F1-skor değerlerinin çok yaklaşık olarak 0 (sıfır) çıkmasına sebep olmaktadır. Verilerin azaltılması ve arttırılması sonrasında elde edilen veri setleri ile 5 hata sınıfı için deneylerin tekrarlanması sonucunda Çizelge 5.2. ile verilen değerler elde edilmiştir. Burada verileri azaltarak dengelemenin olumsuz etkileri görülürken verilerin tekrarlanması ile çoğaltılarak dengelenen veri seti üzerinde başta F1-skor olmak üzere metriklerdeki skorların yükseldiği görülmektedir. Veri seti azaltılarak dengelenmek istendiğinde Özellikle CWE-469 sınıfına ait verilerin azlığı ve diğer sınıflar ile ortak olarak ortaya çıkıyor olması diğer sınıflara ait verilerin neredeyse yok olmasına sebep olmaktadır. Verilerin arttırılması sonucunda diğer sınıflardaki hata sayıları büyük oranda korunduğu için daha dengeli ve güvenilir sonuçlar elde edilebildiği söylenebilir.

## 5.2. Kaynak Kodların Vektörlere Dönüştürülme Yönteminin Etkisi

Kaynak kodların vektörlere dönüştürülmesi benzerlik fonksiyonlarına girdi olarak verilebilmesi için gerekli bir durumdur ancak vektörlere dönüştürülme

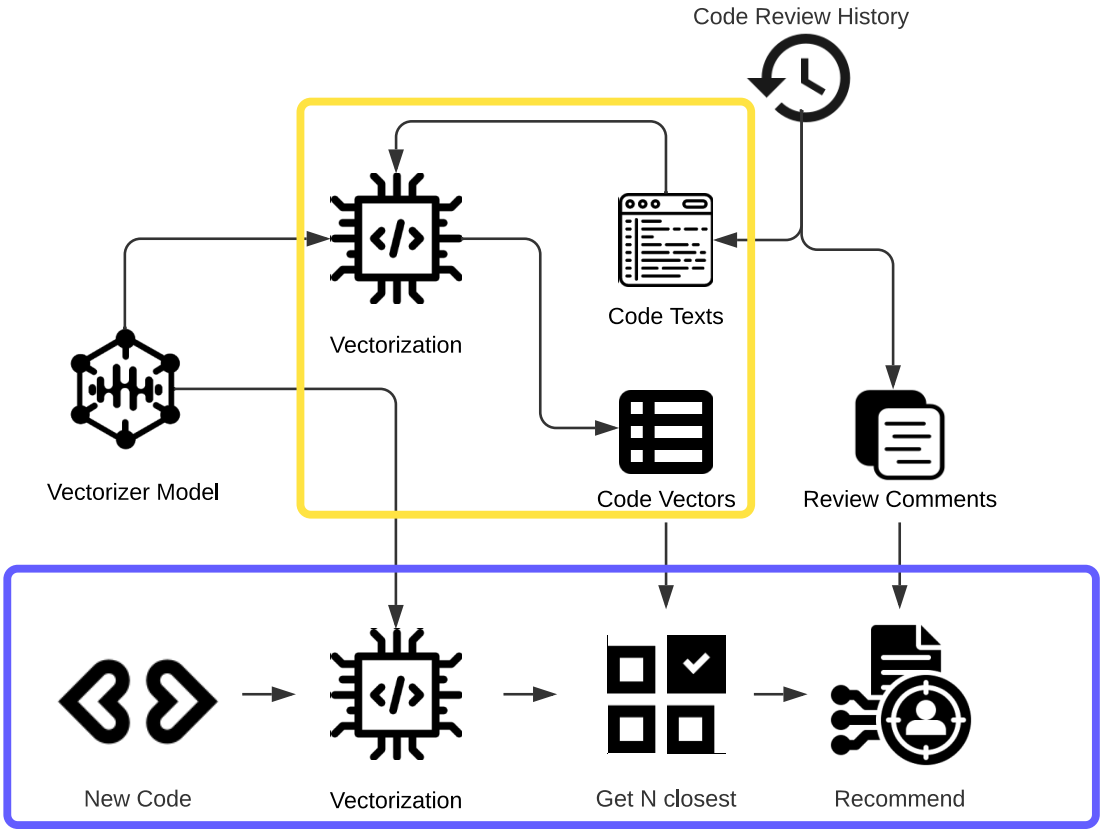
yöntemi ve yaklaşımı benzerlik fonksiyonlarının sergilediği başarıyı etkilemektedir. Daha önce başka çalışmalarca uygulanmış BoW ve TF-IDF yöntemlerine ek olarak ilk defa bu tez kapsamında önerilen Word2Vec, Doc2Vec ve Transformer yöntemleri kaynak kodların vektörlere dönüştürülmesi için kullanılarak Kosinüs, Öklid ve Manhattan uzaklıkları ile çiftler halinde test edilmiştir. Böylece 15 farklı kombinasyon ile sonuçlar Şekil 5.2. ile gösterildiği şekilde elde edilmiştir. Deneyin akışı Şekil 5.3. ile verilmiştir. Sarı çerçeve ile gösterilen akış sadece bir kere uygulanmaktadır. Bu akış ile ilgili detaylar Şekil 5.4. ile verilmektedir. Mor çerçeve ile gösterilen akış bir kaynak kod için öneri istendiğinde sürekli olarak tekrarlanacak adımları içermektedir.




Şekil 5.2. 15 farklı kombinasyonun 3 farklı yöntemle ölçülmesi

Kaynak kodların vektörlere dönüştürülmesinden sorumlu modellerin esnek ve veri değişikliklerine uyarlanabilir olmasını sağlamak için büyük bir kod veri kümesiyle eğitilmesi gerektiğinden Şekil 5.4.'te görüldüğü üzere eğitim için 4,2 milyon java metodu içeren CoDESC veri seti kullanılmıştır. Alsuhaibani vd., 2018; Mnih ve Kavukcuoglu, 2013; Pennington vd., 2014 tarafından yapılan çalışmalarda da görülebileceği gibi, metin gömme performansları korpus boyutu büyüdükçe orantılı olarak artmaktadır. Sonuç olarak, bu boyuttaki verilerle eğitilen vektörleştirici modeller, kaynak kodların vektörel temsillerini daha uygun bir şekilde temsil etmektedir. Aynı zamanda genel bir model oluşturularak yeni veri olması durumunda ortaya çıkacak ince ayar ve yeniden eğitim maliyetlerinin ortadan kaldırılmasına imkan vermektedir.

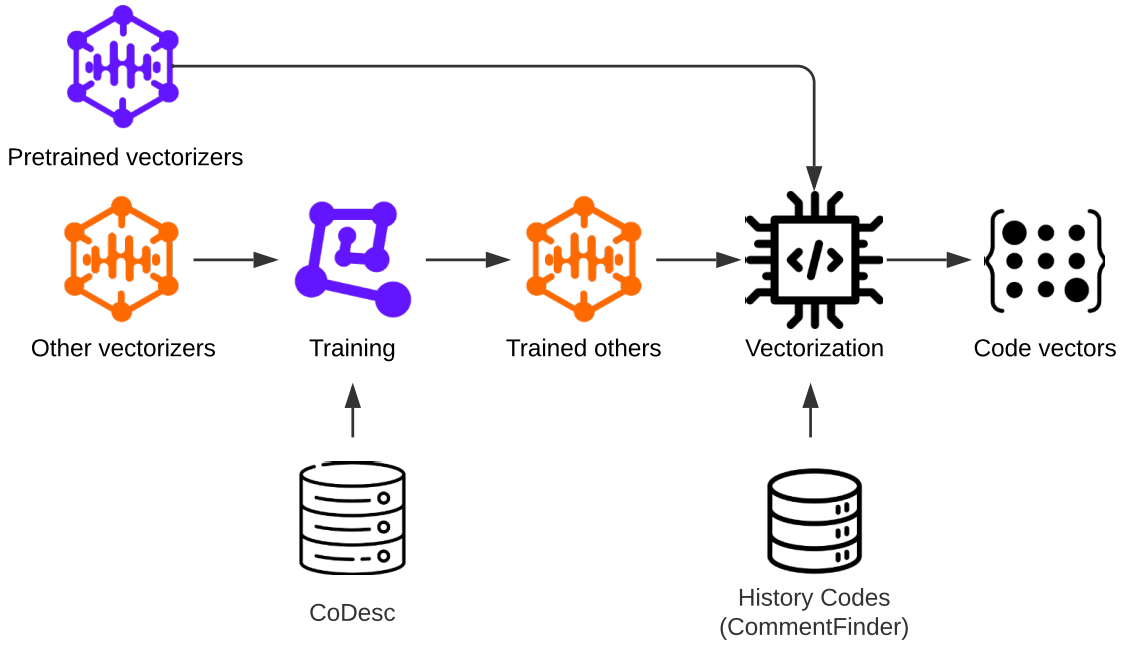
Hem TF-IDF hem de BoW vektöre dönüştürme yöntemleri doğrudan sklearn (Pedregosa vd., 2011) kütüphanesinden varsayılan parametreleriyle kullanılmıştır. Word2Vec ve Doc2Vec yöntemleri ise gensim (Rehurek ve Sojka, 2011) kütüphanesinden Çizelge 5.3. ile verilen parametrelerle eğitilerek




Şekil 5.3. Kod gözden geçirme yorumu önerme adımları

kullanılmıştır. Transformer içinse Guo vd., 2022 tarafından Hugging Face'e (Wolf vd., 2020) yüklenmiş olan ve önceden eğitilmiş transformer yöntemi kullanılmıştır. Bu noktada başka hiçbir ince ayar veya hiperparametre uygulanmamıştır. Tüm yöntemlerin çıktısı olan vektör boyutları Çizelge 5.4. içinde gösterilmiştir.

Öte yandan yapılan deneylerde benzerlik fonksiyonları tarafından Şekil 5.3. içinde verilen kaynak kod gözden geçirme geçmişini temsil edecek Hong vd., 2022 tarafından oluşturulan yaklaşık 150 000 kayıt içeren CommentFinder veri seti kullanılmıştır. Veri seti önceki çalışmada(Hong vd., 2022) 15 000 test ve 135 000 kaynak kod gözden geçirme geçmişi olarak parçalanmıştır. Çapraz doğrulama; modelin gerçek performansını değerlendirmek, overfitting'i tespit etmek, veri kısıtlamalarını telafi etmek için önemli bir araçtır. Makine öğrenimi modellerinin güvenilirlik ve genelleme yeteneğini doğru bir şekilde değerlendirmek için çapraz doğrulama yöntemlerinin kullanılması önemli görünmektedir (A. Ramezan vd., 2019). Bu amaçla bu tez kapsamında 5'li çapraz doğrulama sağlamak için veri seti 5'e bölünmüştür. Böylece her 30 000 kayıt test verisi olarak geriye kalan 120 000 kayıt ise kaynak kod gözden geçirme geçmişi olarak kullanılmıştır. 5 farklı 30 000




Şekil 5.4. Kaynak kodların vektörlere dönüştürülmesi

Çizelge 5.3. Word2Vec ve Doc2Vec parametreleri

| Parametre | Doc2Vec Değeri | Word2Vec Değeri |
|---|---|---|
| vector_size | 300 | 300 |
| window | 15 | 15 |
| min_count | 5 | 10 |
| epochs | 20 | 5 |
| sg | n/a | 1 |
| hs | 0 | 0 |
| dm | 0 | n/a |
| sample | 0,00001 | 0,00001 |
| alpha | 0,03 | 0,03 |
| min_alpha | 0,0007 | 0,0007 |
| workers | 8 | 8 |
| negative | 5 | 5 |

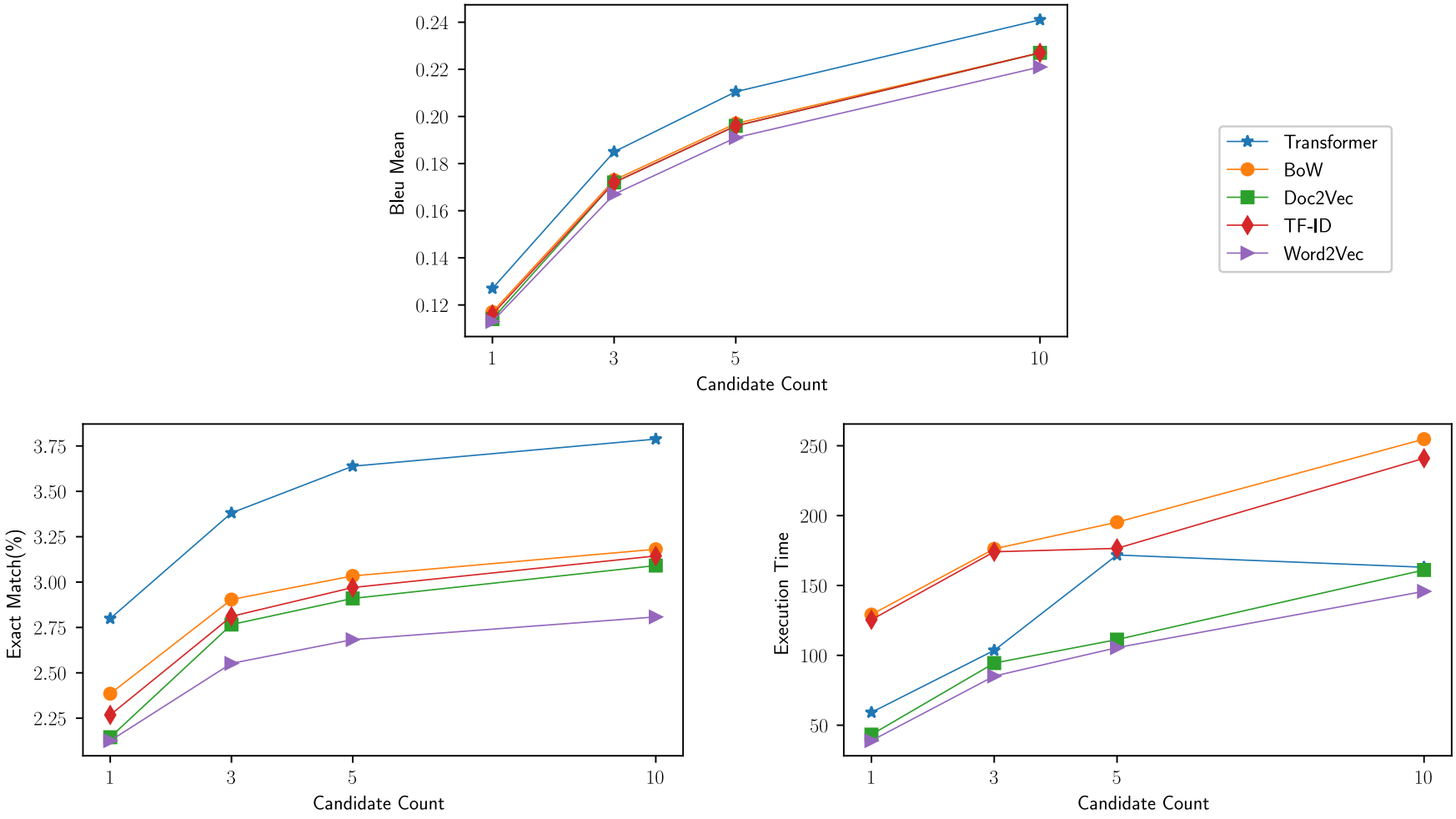
kayıt ve 5 farklı 120 000 kayıt listesi oluşturularak yapılan 5 deneyin ortalaması alınmıştır. Böylece önceki çalışmalara göre 2 kat daha fazla test verisi ve daha doğrulanabilir sonuçlar için 5'li çapraz doğrulama yapılmıştır.

Çizelge 5.4. Vektör boyutları

| Yöntem | Doc2Vec | Word2Vec | Transformer | TF-IDF | BoW |
|---|---|---|---|---|---|
| Vektör Boyutu | 300 | 300 | 768 | 376 565 | 376 565 |

Kurgulanan bu sistem ile kod geçmişinin farklı yazılım ekiplerine göre yeniden uyarlanması ve böylece her ekibin kendi içerisinde oluşturulan kodlama standartlarının dikkate alınması konusunda oldukça esneklik sağlanmıştır.

Kaynak kodların vektörlere dönüştürülmesi ve bahsedilen test veri setlerinin organize edilmesinin ardından benzerliklerin saptanması için bir sonraki adım olan vektör farklarının uygulanması adımı gerçekleştirilmiştir. Burada en yakın k tane komşuluğu (k-nn) bulmak için Kosinüs, Öklit ve Manhattan benzerlik fonksiyonları kullanılmıştır. Buradaki temel mantık en kısa vektör mesafesinin en benzer kaynak kodu temsil etmesi şeklindedir.




Şekil 5.5. Farklı aday öneri sayılarına göre vektörleştirme yöntemlerinin karşılaştırılması

Çizelge 5.5. Vektörleştirme metotlarının tam eşleşme karşılaştırılması

| | Tam Eşleşme (%) | | | |
|---|---|---|---|---|
| Öneri Sayısı | 1 | 3 | 5 | 10 |
| Transformer | **2,799** | **3,381** | **3,638** | **3,788** |
| *BoW* | *2,385* | *2,904* | *3,034* | *3,181* |
| Doc2Vec | 2,145 | 2,766 | 2,910 | 3,091 |
| TF-IDF | 2,268 | 2,811 | 2,970 | 3,144 |
| Word2Vec | 2,126 | 2,552 | 2,683 | 2,808 |

Tahminleri elde etmek için vektör farkları ile hesaplanan mesafeleri sıralayarak, test noktasına en yakın $k$ komşuyu seçtiğimiz *k-nn* (k nearest neighbour) algoritması uygulanmıştır. Burada $k$; 1, 3, 5 ve 10 olarak önceden tanımlanarak verilmiş bir parametredir. Bu tez çalışmasında; kosinüs benzerliği kullanılarak elde edilen önerilerden test verisine en yakın 1, 3, 5 ve 10 aday baz alınarak yapılan önerilere göre vektörize yöntemlerinin karşılaştırılması Çizelge 5.5., Çizelge 5.6. ve Çizelge 5.7. ile verilmiştir. Ayrıca Şekil 5.5. ile grafikler halinde gösterilmektedir. Burada metriklerin hesaplanması için önceki çalışmalarla kıyaslayabilmek adına BLEU skoruna yer verilmiştir.

Çizelge 5.5.'de görüldüğü üzere önerdiğimiz Transformer yöntemi kullanılarak kaynak kodların vektörlere dönüştürülmesi, tam eşleşme ile öneride bulunulması konusunda 1, 3, 5 ve 10 aday öneri olmak üzere tüm aday öneri sayılarında daha yüksek performans göstermiştir. Transformer ile test setindeki kayıtlarda k=1 için %2,799 tam eşleşme önerilmiştir. Bu değer en son çalışma ile verilen BoW ile elde edilen %2,385 değerinden yaklaşık %17 daha iyidir. Bununla birlikte k=10 için transformer ile elde edilen %3,788 değerinin BoW ile elde edilen %3,181 değerinden %19 daha iyi olduğu görülmektedir.

Çizelge 5.6. ile BLEU skorlarının 1, 3, 5 ve 10 aday öneri olmak üzere tüm aday önerileri için de Transformer yöntemiyle daha yüksek elde edildiği görülmektedir. Karşılaştırma için referans alınan çalışmada (Hong vd., 2022) verilen BoW yöntemi ile k=10 için elde edilen 0,227 BLEU skoruna karşılık Transformer yöntemi 0,241 BLEU skoru ile daha iyi performans göstermektedir. Word2Vec yönteminin 0,221 BLEU skoru ile en düşük performansı gösterdiği görülmektedir. BoW, Doc2Vec ve TF-IDF yöntemlerinin tüm $k$ değerleri için çok az farklılıklar gösterse de neredeyse aynı BLEU skorunu verdiği görülmektedir. Bu üç

Çizelge 5.6. Vektörleştirme metotlarının BLEU ortalamasının karşılaştırılması

| | BLEU Ortalaması | | | |
|---|---|---|---|---|
| Öneri Sayısı | 1 | 3 | 5 | 10 |
| Transformer | **0,127** | **0,185** | **0,210** | **0,241** |
| *BoW* | *0,117* | *0,173* | *0,197* | *0,227* |
| Doc2Vec | 0,114 | 0,172 | 0,196 | 0,227 |
| TF-IDF | 0,116 | 0,172 | 0,196 | 0,227 |
| Word2Vec | 0,113 | 0,167 | 0,191 | 0,221 |

Çizelge 5.7. Vektörleştirme metotlarının işlem sürelerine etkisinin karşılaştırılması

| | İşlem Süresi(s) | | | |
|---|---|---|---|---|
| Öneri Sayısı | 1 | 3 | 5 | 10 |
| Transformer | 59 171 | 103 569 | 171 812 | 163 048 |
| *BoW* | *129 201* | *176 234* | *195 284* | *254 778* |
| Doc2Vec | 43 269 | 94 470 | 111 282 | 161 107 |
| TF-IDF | 125 494 | 174 118 | 176 571 | 241 050 |
| Word2Vec | **38 912** | **85 153** | **105 513** | **145 805** |

yöntem ile ilgili olarak özellikle k=10 için 0,227 değeri ile tamamen aynı BLEU skoru elde edilmiştir.

Çizelge 5.7. ile verilen işlem süreleri skorlarının 1, 3, 5 ve 10 aday öneri olmak üzere tüm aday önerileri için de Word2Vec yöntemiyle daha yüksek elde edildiği görülmektedir ancak daha önemli olan diğer BLEU ve tam eşleşme sonuçlarına bakıldığında en düşük skorların Word2Vec ile elde edildiği görülmektedir. İşlem süresi açısından Word2Vec en iyi performansı gösterse de önerdiğimiz Transformer yöntemiyle arasında ihmal edilebilecek bir performans farkı bulunmaktadır. Öte yandan önceki çalışmada önerilen BoW yöntemine göre bu tez kapsamında önerdiğimiz Transformer yöntemi yaklaşık 2 kat daha hızlı sonuçlar üretilmesine sebep olmuştur. Dikkat edilmesi gereken noktalardan birisi buradaki işlem sürelerinin hesaplanması, metin verilerin vektörlere dönüştürülme sürelerinin değil metin verilere karşılık yapılan kod gözden geçirme yorumlarına ait öneri sürelerinin ölçülmesi şeklindedir. Çizelge 5.7. ile verilen değerler 30 000 test verisi için üretilen toplam süreyi ifade etmektedir.

Çizelge 5.8. Uygulanan yöntemlerin ortalama metrik skorları (BLEU, METEOR, ROUGE)

| | | Transformer | BoW | Doc2Vec | TF-IDF | Word2Vec |
|---|---|---|---|---|---|---|
| Kosinüs | Bleu | **0,241** | 0,227 | 0,227 | 0,227 | 0,221 |
| | Meteor | **0,177** | 0,165 | 0,166 | 0,165 | 0,160 |
| | Rouge | **0,266** | 0,254 | 0,253 | 0,253 | 0,249 |
| Öklid | Bleu | **0,239** | 0,220 | 0,217 | 0,225 | 0,221 |
| | Meteor | **0,176** | 0,160 | 0,158 | 0,163 | 0,159 |
| | Rouge | **0,260** | 0,248 | 0,242 | 0,250 | 0,249 |
| Manhattan | Bleu | **0,239** | 0,220 | 0,216 | 0,224 | 0,220 |
| | Meteor | **0,176** | 0,161 | 0,158 | 0,162 | 0,159 |
| | Rouge | **0,260** | 0,248 | 0,242 | 0,248 | 0,249 |

5 farklı vektörleştirme yönteminin (Transformer, BoW, Doc2Vec, TF-IDF, Word2Vec) 3 farklı vektör farkı yöntemi (Kosinüs, Öklid, Manhattan) ile kombinasyonlar halinde kullanıldığı ve tüm kombinasyonlar için 3 farklı değerlendirme metriği (BLEU, METEOR, ROUGE) ile elde edilen bulgular Çizelge 5.8. ile verilmiştir. Çizelge 5.8. ile verilen değerler daha önceki çalışmalar ile kıyaslayabilmek adına 10 aday önerisi için verilen değerleri içermektedir. BLEU, METEOR ve ROUGE ölçüm yöntemlerinin her üçü de önerdiğimiz Transformer yönteminin Kosinüs, Öklid ve Manhattan benzerlik ölçüm yöntemlerinin hepsi ile daha başarılı sonuçlar ortaya koyduğunu göstermektedir. Burada asıl kıyaslama Hong vd., 2022 tarafından önerilen BoW yöntemi ve bu çalışmada önerilen Transformer yöntemi arasında yapılmaktadır. Farklı yaklaşımların ve ölçüm yöntemlerinin çıktılarının görülmesi ve gelecek çalışmalara referans olması amacıyla diğer yöntem ve bu yöntemlerin sonuçlarına da yer verilmiştir. Önceki çalışmalarla karşılaştırıldığında, Transformer yöntemi BLEU skoru için Kosinus yöntemiyle kullanıldığında %6,2'lık bir artış göstermektedir. Kosinüs benzerliği dikkate alındığında BoW, Doc2Vec ve TF-IDF yöntemleri yaklaşık aynı performansı gösterirken Word2Vec en kötü sonuçları vermektedir. Öklid benzerliğinde Transformers hariç diğer vektörleştirici yöntemler benzer sonuçlar göstermektedir yine de ikinci en iyi sonuç TF-IDF ile elde edilmiştir. Manhattan mesafesi de Öklid ile benzer sonuçlar vermektedir. Sonuç itibariyle en iyi performans Kosinüs benzerliği ile edilmiş ve Transformer en yüksek skorların elde edilmesini sağlamıştır. Vektör uzaklıkları için en iyiden en kötüye Kosinüs, Öklid ve Manhattan şeklinde sıralamak doğru olacaktır.

Çizelge 5.9. Uygulanan yöntemlerin tam eşleşme oranları (Yüzde)

| | | Transformer | BoW | Doc2Vec | TF-IDF | Word2Vec |
|---|---|---|---|---|---|---|
| Kosinüs | Bleu=1 | **3,788** | 3,181 | 3,091 | 3,144 | 2,808 |
| | Meteor=1 | **3,769** | 3,170 | 3,084 | 3,133 | 2,797 |
| | Rouge=1 | **3,711** | 3,117 | 3,030 | 3,087 | 2,758 |
| Öklid | Bleu=1 | **3,701** | 2,802 | 2,595 | 2,992 | 2,825 |
| | Meteor=1 | **3,687** | 2,793 | 2,589 | 2,981 | 2,817 |
| | Rouge=1 | **3,620** | 2,754 | 2,549 | 2,937 | 2,783 |
| Manhattan | Bleu=1 | **3,696** | 2,929 | 2,562 | 2,947 | 2,807 |
| | Meteor=1 | **3,683** | 2,919 | 2,552 | 2,940 | 2,798 |
| | Rouge=1 | **3,618** | 2,875 | 2,516 | 2,906 | 2,755 |

5 farklı vektörleştirme yönteminin(Transformer, BoW, Doc2Vec, TF-IDF, Word2Vec) 3 farklı vektör farkı yöntemi (Kosinüs, Öklid, Manhattan) ile kombinasyonlar halinde kullanıldığı ve tüm kombinasyonlar için 3 farklı değerlendirme metriği(BLEU, METEOR, ROUGE) ile elde edilen tam eşleşme oranları Çizelge 5.9. ile verilmiştir. Önerilen transformer yöntemi ile birlikte Kosinüs benzerliği BLEU, METEOR ve ROUGE yöntemlerinin hepsiyle de en iyi performansı göstermektedir.

5 farklı vektörleştirme yönteminin (Transformer, BoW, Doc2Vec, TF-IDF, Word2Vec) 3 farklı vektör farkı yöntemi (Kosinüs, Öklid, Manhattan) ile kombinasyonlar halinde kullanıldığı ve tüm kombinasyonlar için 3 farklı değerlendirme metriği(BLEU, METEOR, ROUGE) ile elde edilen işlem süreleri Çizelge 5.10. ile verilmiştir. Saniye cinsinden verilen değerler 30 000 kayıt için toplam süreyi ifade etmektedir. Farklı benzerlik yöntemleriyle farklı vektörize yöntemlerinin farklı işlem sürelerine sebep olduğu görülmektedir. Önerdiğimiz Transformer yönteminin Manhattan benzerlik yöntemi ile yüksek işlem sürelerine sebep olduğu görülürken Kosinüs benzerlik yöntemi ile ihmal edilebilir bir fark ile en iyi performansı gösteremediği elde edilmiştir. Önerilen sistemin doğru tahmin için elde edilen ölçüm değerleri dikkate alındığında çalışma zamanı süresi Kosinüs benzerliği için dikkate alınmayacak farklar göstermektedir. Öte yandan benzerlik yöntemlerinin işlem sürelerine bakıldığında Kosinüs en verimli, Manhattan ise en az verimli olarak görünmektedir. Özellikle Manhattan benzerlik yöntemi kelime gömme yöntemleri için oldukça düşük performans sergilemektedir.

Çizelge 5.10. Uygulanan yöntemlerin ortalama işlem süreleri (saniye)

| | | Transformer | BoW | Doc2Vec | TF-IDF | Word2Vec |
|---|---|---|---|---|---|---|
| Kosinüs | Bleu | 163,05 | 254,78 | 161,11 | 241,05 | **145,80** |
| | Meteor | 267,53 | 357,10 | 260,12 | 335,74 | **257,73** |
| | Rouge | 148,98 | 215,83 | 144,33 | 249,27 | **139,81** |
| Öklid | Bleu | 234,05 | 260,55 | **193,60** | 243,19 | 206,92 |
| | Meteor | **310,05** | 373,27 | 312,05 | 354,61 | 314,20 |
| | Rouge | 199,61 | 257,41 | **179,05** | 237,32 | 199,69 |
| Manhattan | Bleu | 1942,33 | 284,08 | 1010,25 | **258,21** | 1021,55 |
| | Meteor | 1921,62 | 388,21 | 1081,34 | **362,01** | 1098,42 |
| | Rouge | 1850,43 | 269,47 | 971,73 | **248,72** | 984,03 |

Çizelge 5.11. Çizelge 5.6. ile verilen BLEU skorlarına ait varyans değerleri

| | Varyans | | | |
|---|---|---|---|---|
| Öneri Sayısı | 1 | 3 | 5 | 10 |
| Transformer | 1,20e-06 | 2,02e-06 | 1,69e-06 | 1,67e-06 |
| BoW | 3,14e-06 | 4,23e-06 | 3,26e-06 | 1,84e-06 |
| Doc2Vec | 1,48e-06 | 1,92e-06 | 1,72e-06 | 1,66e-06 |
| TF-IDF | 1,97e-06 | 1,61e-06 | 1,98e-06 | 1,69e-06 |
| Word2Vec | 9,10e-07 | 1,87e-06 | 1,87e-06 | 2,18e-06 |

Bu tez çalışmasında sonuçlar, 5'li çapraz doğrulama tekniğiyle elde edilen 5 değerin ortalaması alınarak sunulmuştur. Çizelge 5.11. ile verilen değerler, Çizelge 5.6. ile verilen BLEU skorlarının varyansını göstermektedir. Çapraz deneyler arasındaki değişkenliğin minimum düzeyde olduğunu ortaya koyan bu veri deneyler sonucu elde edilen bulguların güvenilir olduğunu göstermektedir. Her bir deneme için farklı 30 000 kayıt test verisi olarak kullanıldığından her biri ayrı bir kod gözden geçirme deneyi olarak kabul edilebilir. Bu durumda her bir sonucu elde etmek için beş farklı veri setinin kullanıldığı söylenebilir. Bu bulgular, yöntemin sağlamlığını ve doğruluğunu vurgulamakta ve deneysel tasarımımızın etkinliğine ilişkin fikir vermektedir.

Çizelge 5.12., uygulanan vektöre dönüştürme yöntemleri ve 10 öneri sayısıyla kosinüs benzerliği yaklaşımı kullanan bir kod inceleme öneri modelinin gerçek bir girdiye karşılık önerilen çıktıyı örneklemektedir. Burada elde edilen

Çizelge 5.12. Örnek çıktı

| Kaynak Kod | public String getDirectoryUrl() {<br>return directoryAsyncClient.getDirectoryUrl();<br>} | |
|---|---|---|
| | | Bleu Skoru |
| Actual Comment | This should return a URL | - |
| Transformer | This should return a URL | 1 |
| BoW | Should return a boolean | 0,6433 |
| TF-IDF | Is this a textual description<br>or should this be an enum? | 0,1301 |
| Word2Vec | return queryId | 0,2551 |
| Doc2Vec | Wrap this line | 0,1268 |

örnekte, Transformer yöntemi tarafından sağlanan önerinin gerçek inceleme yorumuyla aynı olduğu görülmektedir. Ek olarak Çizelge 5.12., diğer yöntemlerle sağlanan önerileri ve BLEU skorlarını da göstermektedir.

## 5.3. Sonuç Önerme İçin Eşik Değerinin Etkisi

Yapılan deneylerde 5'li çapraz doğrulama uygulandığı için veri setindeki yaklaşık 150 000 kayıt için en iyi benzerlik skoru hesaplanarak kayıt altına alınmıştır. Bu benzerlik değerleri içinde olabilecek en düşük değerler olan 0 ile en yüksek değer olan 1 de yer almaktadır. Elde edilen bu benzerlik değerleri incelendiğinde çok düşük benzerlik oranlarına sahip olmasına rağmen öneri olarak verilmesinin uygun olmadığı düşünülmüştür. Bu çerçevede bir kaynak kod için en benzer kod aranırken vektör farkları hesaplamasında en yakın vektörün ne kadar yakın olduğu da önemli görünmektedir. Bu kapsamda elde edilen benzerlik skoru için bir eşik değerinin belirlenmesi uygun görünmektedir. Bu eşik değerinin belirlenmesi için öncelikle 1-sigma kuralı ile en yüksek skora sahip ilk %68 kayıt arasından en düşük skora sahip verinin skoru eşik değeri olarak kabul edilmiştir. Benzer olarak benzerlik ve skor arasındaki bağıntıya ait fonksiyonun türevi ile elde edilen pik noktası da eşik değer olarak kabul edilmiştir. Bu deneyde bahsedilen bu benzerlik skoru yani 1-sigma ile eşik değeri **0,5485** olarak, türev ile ise 0.65 olarak elde edilmiştir. Bu şekilde test verisinde bir modelin önerdiği en yakın vektör için benzerlik oranı eşik değerinin altında kalıyorsa öneri yapılmayarak hatalı önerilerin engellenmesi amaçlanmıştır. Böylece 1-sigma ile elde edilen eşik değer referans alındığında BLEU skorunun yeniden hesaplanması sonucunda elde edilen

Çizelge 5.13. Farklı eşik değerlerine göre BLEU skorları

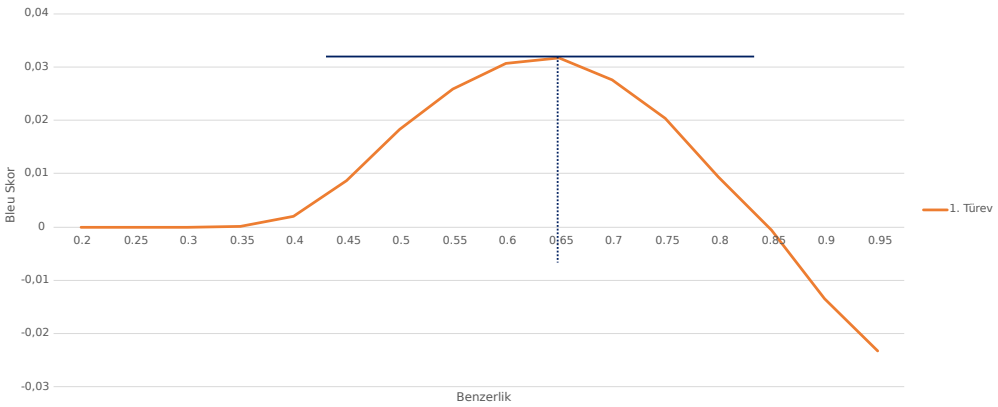
| Etiket | Önerilen Kayıt Sayısı | Eşik Değeri | BLEU Skoru |
|---|---|---|---|
| 1-sigma | 102 461(%68) | 0,5485 | 0,261 |
| 2-sigma | 143 746(%95) | 0,4545 | 0,243 |
| 3-sigma | 150 225(%99,7) | 0,4013 | 0,241 |
| 1. Türev | 59 442(%39,5) | 0,65 | 0,293 |

değer **0,261** şeklindedir. Bu skor sadece Transformer yöntemiyle elde edilen ve en yüksek skor olarak ortaya koyduğumuz *0,241* skorundan daha iyi bir değer olarak sistemde iyileşme sağlamaktadır. Öte yandan türev ile elde edilen eşik değer referans aldındığında ortaya çıkan Bleu skoru *0,293* ile çok daha iyi bir skor sunmaktadır.

Farklı eşik değerlerinin uygulanması durumunda elde edilecek BLEU skorlarına Çizelge 5.13. ile yer verilmiştir. Verilen çizelgede $1\sigma$, $2\sigma$ ve $3\sigma$ ile hesaplanan eşik değerlerinin yanı sıra BLEU skorlarının türevine göre hesaplanan eşik değerine de yer verilmiştir (Şekil 5.6.). Test veri setinde yer alan kayıtlardan 1-sigma ile eşik değer (0,5485) uygulandığında 102 461 kayıt için öneri sunulurken türev ile elde edilen eşik değer (0,65) uygulandığında sadece 59 442 kayıt için öneri sunulmaktadır.

## 5.4. En İyi Öneriyi Yapacak Yöntemin Belirlenmesi

Daha önce de belirtildiği üzere 5 farklı vektörleştirme yöntemi (Transformer, BoW, Doc2Vec, TF-IDF, Word2Vec) 3 farklı vektör farkı yöntemi (Kosinüs, Öklid, Manhattan) ile kombinasyonlar halinde uygulanmıştır. Toplam 15 farklı yöntem uygulandığı görülmektedir. Her ne kadar Kosinüs benzerliği Transformer yöntemi ile birlikte genel ortalamada daha iyi BLEU skorları sağlasa da her bir test verisine tekil olarak bakıldığında farklı yöntemlerin farklı kayıtlar için daha yüksek BLEU skorları ile önerilerde bulunabildiği görülmektedir. Kosinüs benzerliği için örnek veriler Çizelge 5.14. ile verilmiştir. Eğer bir kaynak kod için daha iyi öneri vermesi muhtemel olan yöntem tespit edilip bu yöntemin öneri vermesi sağlanırsa daha doğru önerilerde bulunulabilir. Bu yaklaşımla tüm verilerin etiketlenmesi ve böylece kod gözden geçirme önerisi öncesinde derin öğrenme ile bir sınıflandırma probleminin çözülmesi gerekliliği ortaya çıkmıştır. Veri setinde yer alan tüm veriler tekil olarak en iyi BLEU skoruyla tahminde bulunan yöntem ile etiketlenmiştir. BLEU skorunun aynı olduğu durumlarda işlem sürelerine bakılarak etiket olarak



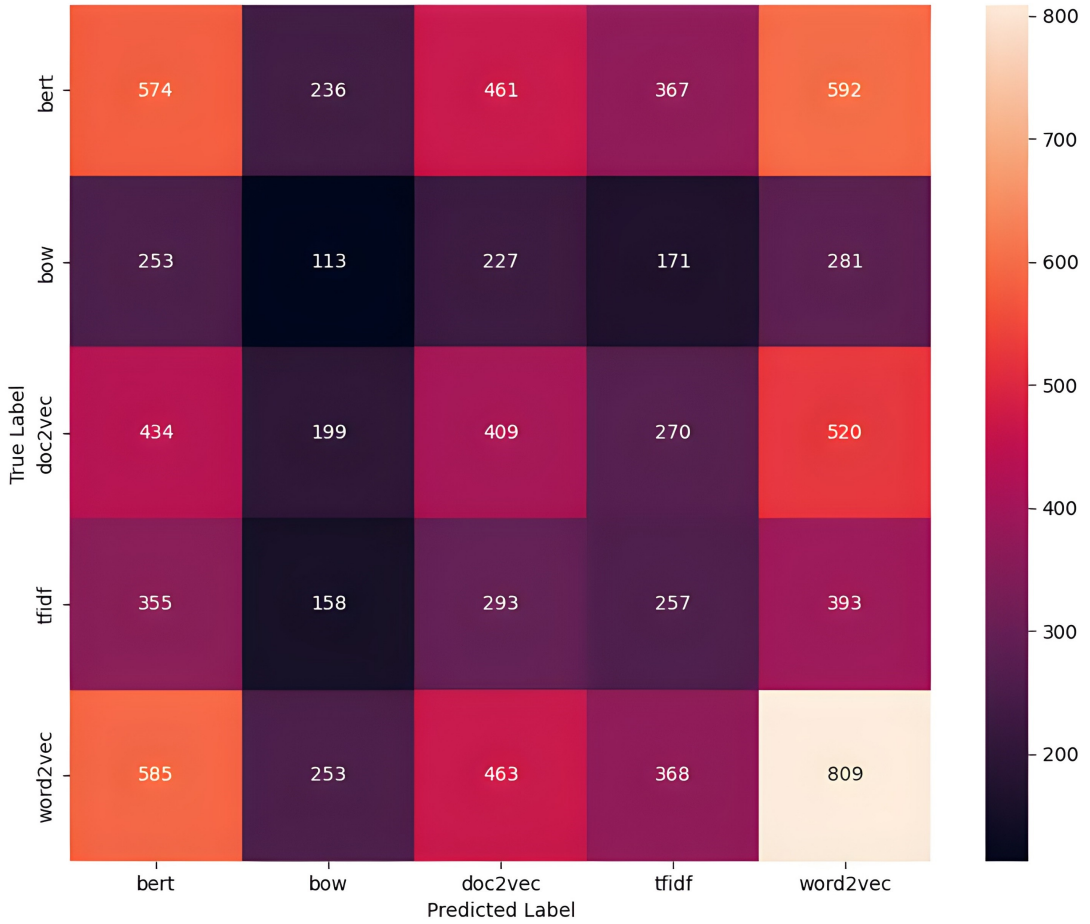

Şekil 5.6. Benzerlik-Bleu skor fonksiyonuna ait türev ile pik noktası

aralarından en hızlı sonuç üreten vektöre dönüştürme yöntemi seçilmiştir. Bu şekilde 5 sınıflı bir veri seti elde edilmiştir.

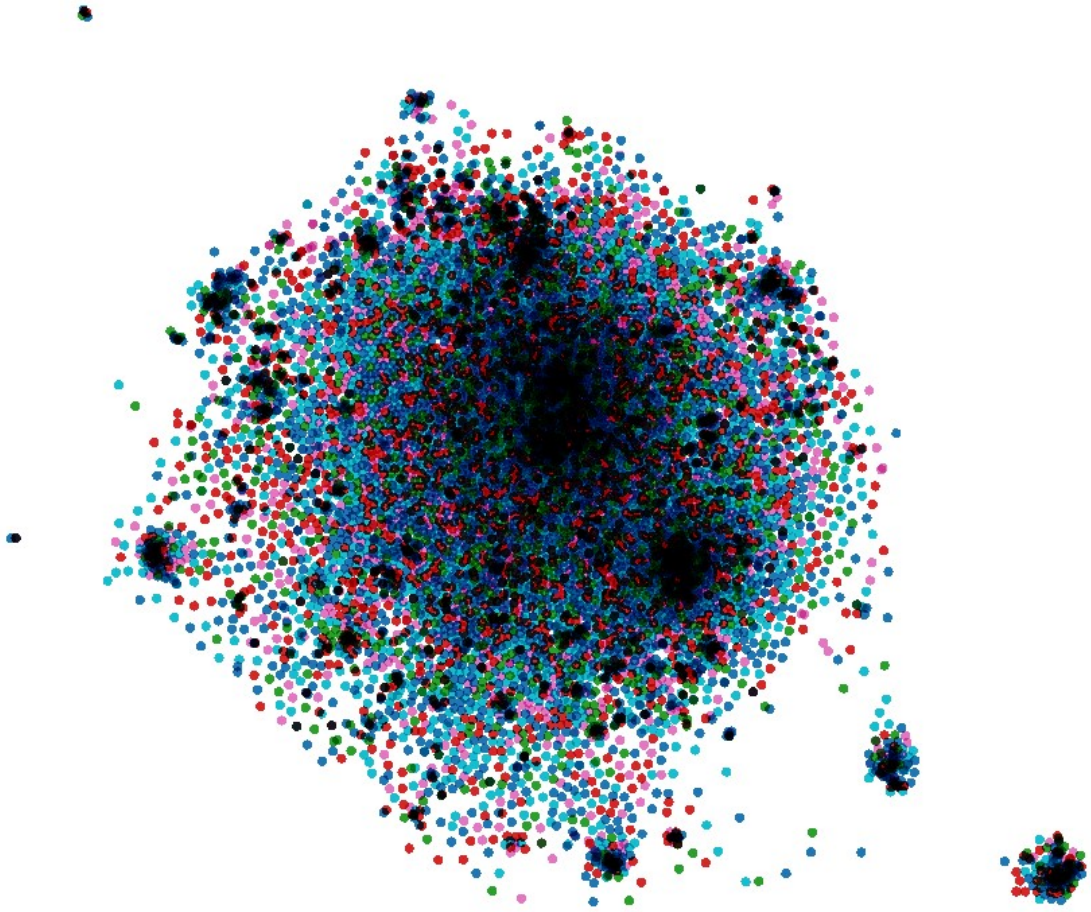
Çizelge 5.14. Tekil olarak daha iyi BLEU skoruna sahip vektöre dönüştürme yöntemlerine ait örnek kayıtlar

| İndeks | Transformer | BoW | Doc2Vec | TF-IDF | Word2Vec |
|---|---|---|---|---|---|
| 119725 | 0,1667 | 0,2746 | **0,3150** | 0,2834 | 0,2133 |
| 4250 | 0,1700 | **0,1894** | 0,1698 | 0,1214 | 0,1243 |
| 54619 | 0,2165 | 0,2479 | 0,1934 | 0,2258 | **0,3119** |
| 17304 | 0,3021 | 0,3021 | 0,3152 | **0,3371** | 0,3021 |

Elde edilen veri seti ile yapılan deneye ait sonuçlar BLEU skorlarında ters etkiye sebep olmuştur. *Annoy* kullanılarak elde edilen sonuçlara bakıldığında hata matrisinde (Şekil 5.7.) sınıflara ait önerilerin çok dağınık olduğu görülmektedir. Veri setinde yer alan metin verilerin benzerlik ve ilişkilerini daha iyi gözlemleyebilmek için verilerin T-SNE yöntemiyle görselleştirilmesi sağlanmıştır. Bu görselleştirme sonucunda Şekil 5.8.'de görüldüğü üzere 2 boyutlu uzayda homojen dağılmış noktalar kaynak kod benzerliklerinin tespitinin zorlu bir problem olduğunu göstermektedir.




Şekil 5.7. Sınıflandırılan veriler üzerinde yapılan çalışmaya ait hata matrisi



Şekil 5.8. Sınıflandırılmış veri setinde yer alan kaynak kodların T-SNE ile görselleştirilmiş grafiği (iterasyon: 5000, perplexity: 25, learning rate: 10)

# 6. SONUÇ VE ÖNERİLER

Yazılım geliştirme projelerinde kod kalitesini arttırırken hata sayısının azaltılması, geliştirilen bir kodun başka bakış açılarıyla incelenmesi ve gerekli durumlarda organize bir şekilde yeniden düzenlenmesi için yapılan kod gözden geçirme faaliyetleri uzun yıllardır çeşitli şekillerde ele alınmaya çalışılan ve en iyi pratikler arasında gösterilen bir yazılım geliştirme süreci adımıdır.

Günümüzde standartlaşan konfigürasyon yönetim araçları ile bir yazılıma ait geliştirilen tüm kaynak kodlara ait değişikliklerin takip edilmesi ve geliştiriciler tarafından henüz geliştirme aşamasındayken bile bağımsız olarak incelenmesi mümkündür. Özellikle bir proje ekibine yeni dahil olan geliştiricilerin hem kurum standartlarını uygularken yapabileceği hataların önüne geçilmesinde hem de teknik olarak bilginin diğer geliştiriclere aktarılmasında konfigürasyon araçlarının sunduğu kod gözden geçirme imkanları hem özel projelerde hem de açık kaynak kodlu projelerde bir kural olarak uygulanmaktadır.

Uzun yıllardır özellikle açık kaynak kodlu projelerde biriken kod gözden geçirme yorumlarının varlığı buradaki bilgiyle yapay öğrenme modellerinin eğitilebilmesine imkan sağlamaktadır. Buna rağmen yazılım geliştirme alanında yapılan bir çok çalışma arasında kod gözden geçirme üzerine yapılan çalışmaların henüz tam olgunlaşmamış ve kısıtlı sayıda olduğu da yapılan tespitler arasındadır.

Verinin ön işlenmesi yapay öğrenme modellerinde çeşitli şekillerde ve sıklıkla yapılan bir adımdır. Bazı durumlarda daha hızlı sonuçlar üretmek için bazı durumlarda yapay öğrenme modelinin daha iyi eğitilebilmesi için çoğunlukla ham veri üzerine doğrudan uygulanmaktadır. Örneğin kaynak kod üzerinde yapılan yorumlarda "Güzel iş, tebrikler" gibi teknik olarak anlamsız kayıtların temizlenmesi gerektiği gibi verinin içerisinde geçen ve teknik bilgiyi doğrudan içermeyen "degisken1, degisken2" gibi geliştiricinin inisiyatifiyle farklılık gösterebilecek değişken ya da parametre isimlerinin öğrenmeye etkilerinin uygun şekilde en aza indirgenmesi gerekebilmektedir.

Teknolojinin hızla ilerlemesinde donanımla birlikte yazılımın yani bilgisayar destekli sistemlerin katkısı yadsınamaz boyuttadır. Ancak teknolojinin bu hızla ilerlemesi diğer bir çok alandaki bilgilerin hızla farklı şekillere evrilmesine ya da

uygulanan pratiklerin değiştirilmesine sebep olurken yazılım geliştirme süreçlerinin ve hatta programlama dillerindeki en iyi pratiklerin bile değişmesine sebep olabilmektedir. Bu açıdan düşünüldüğünde kod gözden geçirme yapan geliştiricilerin bakış açılarında ve özellikle güvenlik konusundaki yaklaşımlarında da değişiklikler olabilmektedir. Bu sebeple sürekli öğrenen ve mevcut verilerden beslenen bir yaklaşım, büyük ve küçük çaplı tüm yazılım ekipleri tarafından gündemde olan bir problemi çözebilecek şekilde görülmektedir.

Bu tez kapsamında kaynak kod gözden geçirme yorumlarının en doğru şekilde önerilmesi için mevcut literatüre katkı sağlayacak tekniklerin yenilikçi ve özgün bir şekilde uygulanması ve yapılan bu çalışmaların sonucunun tekrarlanabilir, sağlam ve doğrulanabilir olarak paylaşılması için çalışmalar yürütülmüştür.

Kaynak kodlardaki mantıksal ve sözdizimsel ilişkinin yakalanabilmesi amacıyla sözcüksel analiz işleminin tahminleme başarısına etkisini test etmek amacıyla yürütülen çalışmada sözcüksel analiz işleminden geçirilmiş veriler ile oluşturulan modelin tahminleme başarısını artırdığı gözlenmiştir. Buradan yola çıkarak kaynak kodların değişken türleri, değişken isimleri, fonksiyonlar, yorum satırları, boşluklar ve özel karakterlerden ayrıştırılarak kök haline getirilmesinin modelin hata tahminleme oranında olumlu etkisinin olduğu söylenebilir. Bu çalışmadan elde edilen diğer bir çıktı ise kullanılan DRAPER VDISC veri setinin dengesiz veriler içeriyor olmasıdır.

Veri ön işlemeden sonra verilerin vektörlere dönüştürülmesi aşamasında önceki çalışmalarda test edilmiş olan TF-IDF ve Bow yöntemlerine ek olarak Word2Vec, Doc2Vec ve Transformer yöntemleri de dahil edilmiş ve bu vektörleştirme yöntemleri Kosinüs, Öklid ve Manhattan vektör farkı ölçme tekniklerinin her biriyle test edilerek sonuçlar elde edilmiştir. Bulgular, Transformer ile vektörlere dönüştürülen veriler üzerinde Kosinüs benzerliğinin diğer tüm yaklaşımlardan daha yüksek BLEU, METEOR ve ROUGE skorları elde ettiğini göstermektedir. Önerdiğimiz yöntemlerden Transformer yöntemi BLEU skoruna göre literatürdeki en iyi çalışmadan %6,5, tam eşleşme skoruna göre ise %19,1 daha iyi performans göstermiştir.

Elde edilen sonuçların önerilmesinde en yüksek benzerlik skoru için eşik değerinin belirlenmesi sağlanmıştır. Kullanılan istatistiksel yaklaşım sonucunda elde edilen eşik değerinin uygulandığı önerme deneylerinde bu tez çalışmasında yapılan önerileri BLEU skoruna göre %8,2 daha yükseltmiştir. Böylece literatürdeki

en iyi çalışmada verilen BLEU skor sonuçlarına göre %14,9 daha iyi performans elde edilmiştir.

Gelecekte yapılacak çalışmalar, bu çalışmanın tüm bulgularını tekrarlayabilecek ve kendi yöntemleri ile karşılaştırabilecektir. Geliştirilecek bir uygulama ile gerçek kullanıcı geri dönüşleriyle(iyi/orta/kötü) doğruluk oranlarının ölçülmesi ve bu geri dönüşlerin sistemi beslemesi ile daha doğru sonuçlar önerilmesi de planlanan bir çalışmadır.

# KAYNAKLAR DİZİNİ

A. Ramezan, C., A. Warner, T., & E. Maxwell, A. (2019). Evaluation of sampling and cross-validation tuning strategies for regional-scale machine learning classification. *Remote Sensing*, *11*(2), 185.

Al Qasem, O., Akour, M., & Alenezi, M. (2020). The influence of deep learning algorithms factors in software fault prediction. *IEEE Access*, *8*, 63945–63960.

Aleem, S., Capretz, L. F., Ahmed, F., vd., (2015). Comparative performance analysis of machine learning techniques for software bug detection. *Proceedings of the 4th International Conference on Software Engineering and Applications*, (1), 71–79.

Allamanis, M., Barr, E. T., Bird, C., & Sutton, C. (2015). Suggesting accurate method and class names. *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, 38–49.

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, *51*(4), 1–37.

Alsolai, H., & Roper, M. (2020). A systematic literature review of machine learning techniques for software maintainability prediction. *Information and Software Technology*, *119*, 106214.

Alsuhaibani, M., Bollegala, D., Maehara, T., & Kawarabayashi, K.-i. (2018). Jointly learning word embeddings using a corpus and a knowledge base. *PloS one*, *13*(3), e0193094.

Amorim, L., Costa, E., Antunes, N., Fonseca, B., & Ribeiro, M. (2015). Experience report: Evaluating the effectiveness of decision trees for detecting code smells. *2015 IEEE 26th international symposium on software reliability engineering (ISSRE)*, 261–269.

Aniche, M., Maziero, E., Durelli, R., & Durelli, V. H. (2020). The effectiveness of supervised machine learning algorithms in predicting software refactoring. *IEEE Transactions on Software Engineering*, *48*(4), 1432–1450.

# KAYNAKLAR DİZİNİ

Aubaid, A. M., & Mishra, A. (2020). A rule-based approach to embedding techniques for text document classification. *Applied Sciences*, *10*(11), 4009.

Axelsson, S., Baca, D., Feldt, R., Sidlauskas, D., & Kacan, D. (2009). Detecting defects with an interactive code review tool based on visualisation and machine learning. *the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009)*.

Azeem, M. I., Palomba, F., Shi, L., & Wang, Q. (2019). Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology*, *108*, 115–138.

Baca, D., Carlsson, B., & Lundberg, L. (2008). Evaluating the cost reduction of static code analysis for software security. *Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security*, 79–88.

Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. *2013 35th International Conference on Software Engineering (ICSE)*, 712–721.

Banerjee, S., & Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 65–72.

Barki, H., Rivard, S., & Talbot, J. (1993). Toward an assessment of software development risk. *Journal of management information systems*, *10*(2), 203–225.

Baum, T., Liskin, O., Niklas, K., & Schneider, K. (2016a). A faceted classification scheme for change-based industrial code review processes. *2016 IEEE International conference on software quality, reliability and security (QRS)*, 74–85.

Baum, T., Liskin, O., Niklas, K., & Schneider, K. (2016b). Factors influencing code review processes in industry. *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering*, 85–96.

# KAYNAKLAR DİZİNİ

Bergmann, S. D. (2003). Compilers.

Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., & Engler, D. (2010). A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM*, *53*(2), 66–75.

Bhandari, G. P., & Gupta, R. (2018). Machine learning based software fault prediction utilizing source code metrics. *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, 40–45.

Braga, R., Neto, P. S., Rabêlo, R., Santiago, J., & Souza, M. (2018). A machine learning approach to generate test oracles. *Proceedings of the XXXII Brazilian Symposium on Software Engineering*, 142–151.

Butgereit, L. (2019). Using machine learning to prioritize automated testing in an agile environment. *2019 Conference on Information Communications Technology and Society (ICTAS)*, 1–6.

Cetiner, M., & Sahingoz, O. K. (2020). A comparative analysis for machine learning based software defect prediction systems. *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1–7.

Chen, Z., & Monperrus, M. (2019). A literature study of embeddings on source code. *arXiv preprint arXiv:1904.03061*.

Chess, B., & McGraw, G. (2004). Static analysis for security. *IEEE security & privacy*, *2*(6), 76–79.

Choudhary, G. R., Kumar, S., Kumar, K., Mishra, A., & Catal, C. (2018). Empirical analysis of change metrics for software fault prediction. *Computers & Electrical Engineering*, *67*, 15–24.

# KAYNAKLAR DİZİNİ

Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th international conference on Machine learning*, 160–167.

Cui, J., Wang, L., Zhao, X., & Zhang, H. (2020). Towards predictive analysis of android vulnerability using statistical codes and machine learning for IoT applications. *Computer Communications*, *155*, 125–131.

Cunha, A., Conte, T., & Gadelha, B. (2021). Code Review is just reviewing code? A qualitative study with practitioners in industry. *Brazilian Symposium on Software Engineering*, 269–274.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dhamayanthi, N., & Lavanya, B. (2019). Improvement in software defect prediction outcome using principal component analysis and ensemble machine learning algorithms. *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*, 397–406.

Du, X., Chen, B., Li, Y., Guo, J., Zhou, Y., Liu, Y., & Jiang, Y. (2019). Leopard: Identifying vulnerable code for vulnerability assessment through program metrics. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 60–71.

Fan, G., Diao, X., Yu, H., Yang, K., & Chen, L. (2019). Deep semantic feature learning with embedded static metrics for software defect prediction. *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, 244–251.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., vd., (2020). Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

# KAYNAKLAR DİZİNİ

Gopalakrishnan, R., Sharma, P., Mirakhorli, M., & Galster, M. (2017). Can latent topics in source code predict missing architectural tactics? *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 15–26.

Goues, C. L., Pradel, M., & Roychoudhury, A. (2019). Automated program repair. *Communications of the ACM*, *62*(12), 56–65.

Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., & Yin, J. (2022). Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850*.

Gupta, A. [Aakanshi], Suri, B., Kumar, V., & Jain, P. (2021). Extracting rules for vulnerabilities detection with static metrics using machine learning. *International Journal of System Assurance Engineering and Management*, *12*, 65–76.

Gupta, A. [Anshul], & Sundaresan, N. (2018). Intelligent code reviews using deep learning. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18) Deep Learning Day*.

Harer, J. A., Kim, L. Y., Russell, R. L., Ozdemir, O., Kosta, L. R., Rangamani, A., Hamilton, L. H., Centeno, G. I., Key, J. R., Ellingwood, P. M., vd., (2018). Automated software vulnerability detection with machine learning. *arXiv preprint arXiv:1803.04497*.

Hasan, M., Muttaqueen, T., Ishtiaq, A. A., Mehrab, K. S., Haque, M. M. A., Hasan, T., Ahmad, W. U., Iqbal, A., & Shahriyar, R. (2021). Codesc: A large code-description parallel dataset. *arXiv preprint arXiv:2105.14220*.

Havrlant, L., & Kreinovich, V. (2017). A simple probabilistic explanation of term frequency-inverse document frequency (tf-idf) heuristic (and variations motivated by this explanation). *International Journal of General Systems*, *46*(1), 27–36.

# KAYNAKLAR DİZİNİ

Hellendoorn, V. J., & Devanbu, P. (2017). Are deep neural networks the best choice for modeling source code? *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 763–773.

Hong, Y., Tantithamthavorn, C., Thongtanunam, P., & Aleti, A. (2022). CommentFinder: a simpler, faster, more accurate code review comments recommendation. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 507–519.

Hu, G., Zhu, L., & Yang, J. (2018). AppFlow: using machine learning to synthesize robust, reusable UI tests. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 269–282.

Huizinga, D., & Kolawa, A. (2007). *Automated defect prevention: best practices in software management*. John Wiley & Sons.

Jones, C. (2017). *The Mess of Software Metrics* [Erişim tarihi: 01.06.2023]. https://namcook.com/articles/The%5C%20Mess%5C%20of%5C%20Software%5C%20Metrics%5C%202017.pdf

Kabak, M., Sağlam, F., & Aktaş, A. (2017). Farklı uzaklık hesaplama yaklaşımlarının TOPSIS üzerinde kullanılabilirliğinin incelenmesi. *Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi*.

Kanade, A., Maniatis, P., Balakrishnan, G., & Shi, K. (2020). Learning and evaluating contextual embedding of source code. *International conference on machine learning*, 5110–5121.

Kartal, Y., Ceyhan, M., & Özkan, K. (2021). Derin Öğrenme ile Yazılım Hata Tespiti. In M. Gök (Ed.), *International Conference on Data Science and Applications (ICONDATA'21)* (pp. 230–234).

# KAYNAKLAR DİZİNİ

Kaur, A., Jain, S., & Goel, S. (2017). A support vector machine based approach for code smell detection. *2017 International Conference on Machine Learning and Data Science (MLDS)*, 9–14.

Kim, H. W. (2020). A Study on the Application of Agile Methodology to Improve Software Development Quality. *International journal of advanced smart convergence*, *9*(3), 59–70.

Kosker, Y., Turhan, B., & Bener, A. (2009). An expert system for determining candidate software classes for refactoring. *Expert Systems with Applications*, *36*(6), 10000–10003.

Kumar, L., & Sureka, A. (2017). Application of LSSVM and SMOTE on seven open source projects for predicting refactoring at class level. *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 90–99.

Lal, H., & Pahwa, G. (2017). Code review analysis of software system using machine learning techniques. *2017 11th International Conference on Intelligent Systems and Control (ISCO)*, 8–13.

Le, T. H., Chen, H., & Babar, M. A. (2020). Deep learning for source code modeling and generation: Models, applications, and challenges. *ACM Computing Surveys (CSUR)*, *53*(3), 1–38.

LeClair, A., Jiang, S., & McMillan, C. (2019). A neural model for generating natural language summaries of program subroutines. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 795–806.

Li, L., Yang, L., Jiang, H., Yan, J., Luo, T., Hua, Z., Liang, G., & Zuo, C. (2022). AUGER: automatically generating review comments with pre-training models. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1009–1021.

# KAYNAKLAR DİZİNİ

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., & Lin, X. (2019). Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, *32*(8), 1475–1488.

Li, Z., Lu, S., Guo, D., Duan, N., Jannu, S., Jenks, G., Majumder, D., Green, J., Svyatkovskiy, A., Fu, S., vd., (2022). Automating code review activities by large-scale pre-training. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1035–1047.

Lima, R., da Cruz, A. M. R., & Ribeiro, J. (2020). Artificial intelligence applied to software testing: A literature review. *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, 1–6.

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. *Text summarization branches out*, 74–81.

Liu, F., Li, G., Zhao, Y., & Jin, Z. (2020). Multi-task learning based pre-trained language model for code completion. *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 473–485.

Liu, Z., Xia, X., Hassan, A. E., Lo, D., Xing, Z., & Wang, X. (2018). Neural-machine-translation-based commit message generation: how far are we? *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 373–384.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., vd., (2021). Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*.

Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

# KAYNAKLAR DİZİNİ

Martin, B., Brown, M., Paller, A., Kirby, D., & Christey, S. (2011). CWE. *SANS top*, *25*.

Martin, B., Brown, M., Paller, A., Kirby, D., & Christey, S. [Steve]. (2011). 2011 CWE/SANS top 25 most dangerous software errors. *Common Weakness Enumeration*, *7515*.

Martin, R. A., & Barnum, S. (2008). Common weakness enumeration (cwe) status update. *ACM SIGAda Ada Letters*, *28*(1), 88–91.

Mnih, A., & Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. *Advances in neural information processing systems*, *26*.

Murali, V., Chaudhuri, S., & Jermaine, C. (2017). Bayesian specification learning for finding API usage errors. *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 151–162.

Nguyen, D.-M., Do, H.-N., Huynh, Q.-T., Vo, D.-T., & Ha, N.-H. (2019). Shinobi: A novel approach for context-driven testing (CDT) using heuristics and machine learning for web applications. *Industrial Networks and Intelligent Systems: 14th EAI International Conference, INISCOM 2018, Da Nang, Vietnam, August 27–28, 2018, Proceedings*, 86–102.

Omri, S., & Sinz, C. (2020). Deep learning for software defect prediction: A survey. *Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops*, 209–214.

Pan, C., Lu, M., & Xu, B. (2021). An empirical study on software defect prediction using codebert model. *Applied Sciences*, *11*(11), 4793.

Pantiuchina, J., Bavota, G., Tufano, M., & Poshyvanyk, D. (2018). Towards just-in-time refactoring recommenders. *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 312–3123.

# KAYNAKLAR DİZİNİ

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 311–318.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., vd., (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, 2825–2830.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Piskachev, G., Do, L. N. Q., & Bodden, E. (2019). Codebase-adaptive detection of security-relevant methods. *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 181–191.

Pradel, M., & Sen, K. (2018). Deepbugs: A learning approach to name-based bug detection. *Proceedings of the ACM on Programming Languages*, *2*(OOPSLA), 1–25.

Qader, W. A., Ameen, M. M., & Ahmed, B. I. (2019). An overview of bag of words; importance, implementation, applications, and challenges. *2019 international engineering conference (IEC)*, 200–204.

Qiao, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning based software defect prediction. *Neurocomputing*, *385*, 100–110.

Rathore, S. S., & Kumar, S. (2021). Software fault prediction based on the dynamic selection of learning technique: findings from the eclipse project study. *Applied Intelligence*, 1–16.

Rehurek, R., & Sojka, P. (2011). Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, *3*(2), 2.

# KAYNAKLAR DİZİNİ

Russell, R., Kim, L., Hamilton, L., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P., & McConley, M. (2018). Automated vulnerability detection in source code using deep representation learning. *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, 757–762.

Sayyad Shirabad, J., & Menzies, T. (2005). The PROMISE Repository of Software Engineering Databases. [Erişim tarihi: 01.06.2023]]. http://promise.site.uottawa.ca/SERepository

Shabtai, A., Moskovitch, R., Elovici, Y., & Glezer, C. (2009). Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *information security technical report*, *14*(1), 16–29.

Sharma, T., Kechagia, M., Georgiou, S., Tiwari, R., Vats, I., Moazen, H., & Sarro, F. (2021). A survey on machine learning techniques for source code analysis. *arXiv preprint arXiv:2110.09610*.

Shen, Z., & Chen, S. (2020). A survey of automatic software vulnerability detection, program repair, and defect prediction techniques. *Security and Communication Networks*, *2020*, 1–16.

Shuai, J., Xu, L., Liu, C., Yan, M., Xia, X., & Lei, Y. (2020). Improving code search with co-attentive representation learning. *Proceedings of the 28th International Conference on Program Comprehension*, 196–207.

Singh, A. K., & Shashi, M. (2019). Vectorization of text documents for identifying unifiable news articles. *International Journal of Advanced Computer Science and Applications*, *10*(7).

Siow, J. K., Gao, C., Fan, L., Chen, S., & Liu, Y. (2020). Core: Automating review recommendation for code changes. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 284–295.

# KAYNAKLAR DİZİNİ

Song, Q., Guo, Y., & Shepperd, M. (2018). A comprehensive investigation of the role of imbalanced learning for software defect prediction. *IEEE Transactions on Software Engineering*, *45*(12), 1253–1269.

Sotto-Mayor, B., & Kalech, M. (2021). Cross-project smell-based defect prediction. *Soft Computing*, *25*(22), 14171–14181.

Tanwar, A., Sundaresan, K., Ashwath, P., Ganesan, P., Chandrasekaran, S. K., & Ravi, S. (2020). Predicting vulnerability in large codebases with deep code representation. *arXiv preprint arXiv:2004.12783*.

Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., & Matsumoto, K.-i. (2015). Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 141–150.

Tufano, M., Watson, C., Bavota, G., Di Penta, M., White, M., & Poshyvanyk, D. (2018). Deep learning similarities from different representations of source code. *Proceedings of the 15th international conference on mining software repositories*, 542–553.

Tufano, M., Watson, C., Bavota, G., Di Penta, M., White, M., & Poshyvanyk, D. (2019). Learning how to mutate source code from bug-fixes. *2019 IEEE International conference on software maintenance and evolution (ICSME)*, 301–312.

Tufano, M., Watson, C., Bavota, G., Penta, M. D., White, M., & Poshyvanyk, D. (2019). An empirical study on learning bug-fixing patches in the wild via neural machine translation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *28*(4), 1–29.

Tufano, R., Masiero, S., Mastropaolo, A., Pascarella, L., Poshyvanyk, D., & Bavota, G. (2022). Using pre-trained models to boost code review

# KAYNAKLAR DİZİNİ

automation. *Proceedings of the 44th International Conference on Software Engineering*, 2291–2302.

Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D., & Bavota, G. (2021). Towards automating code review activities. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 163–174.

Turner, C. A., Jacobs, A. D., Marques, C. K., Oates, J. C., Kamen, D. L., Anderson, P. E., & Obeid, J. S. (2017). Word2Vec inversion and traditional text classifiers for phenotyping lupus. *BMC medical informatics and decision making*, *17*(1), 1–11.

Ucci, D., Aniello, L., & Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers & Security*, *81*, 123–147.

Uchoa, A., Barbosa, C., Oizumi, W., Blenilio, P., Lima, R., Garcia, A., & Bezerra, C. (2020). How does modern code review impact software design degradation? an in-depth empirical study. *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 511–522.

Utting, M., Legeard, B., Dadeau, F., Tamagnan, F., & Bouquet, F. (2020). Identifying and generating missing tests using machine learning on execution traces. *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 83–90.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, *9*(11).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Vikipedi. (2022). Sözcüksel analiz [Erişim tarihi: 01.06.2023]. https://en.wikipedia.org/wiki/Lexical_analysis

# KAYNAKLAR DİZİNİ

Vikipedi. (2023). *Normal dağılım* [Erişim Tarihi: 01.06.2023]. https://tr.wikipedia.org/wiki/Normal_da%C4%9F%C4%B1l%C4%B1m

Wang, S. [Shuai], Liu, J., Qiu, Y., Ma, Z., Liu, J., & Wu, Z. (2019). Deep learning based code completion models for programming codes. *Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control*, 1–9.

Wang, S. [Song], Chollak, D., Movshovitz-Attias, D., & Tan, L. (2016). Bugram: bug detection with n-gram language models. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 708–719.

Wattenberg, M., Viégas, F., & Johnson, I. (2016). How to use t-SNE effectively. *Distill*, *1*(10), e2. https://doi.org/10.23915/distill.00002

White, M., Tufano, M., Martinez, M., Monperrus, M., & Poshyvanyk, D. (2019). Sorting and transforming program repair ingredients via deep learning code similarities. *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 479–490.

White, M., Tufano, M., Vendome, C., & Poshyvanyk, D. (2016). Deep learning code fragments for code clone detection. *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, 87–98.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., vd., (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 38–45.

Xu, J., Wang, F., & Ai, J. (2020). Defect prediction with semantics and context features of codes based on graph representation learning. *IEEE Transactions on Reliability*, *70*(2), 613–625.

Yahav, E. (2018). From programs to interpretable deep models and back. *Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the*

# KAYNAKLAR DİZİNİ

*Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I 30*, 27–37.

Yang, H., Li, S., Wu, X., Lu, H., & Han, W. (2019). A novel solutions for malicious code detection and family clustering based on machine learning. *IEEE Access*, *7*, 148853–148860.

Yosifova, V., Tasheva, A., & Trifonov, R. (2021). Predicting vulnerability type in common vulnerabilities and exposures (CVE) database with machine learning classifiers. *2021 12th National Conference with International Participation (ELECTRONICA)*, 1–6.

Zhang, H., & Zhou, L. (2019). Similarity judgment of civil aviation regulations based on Doc2Vec deep learning algorithm. *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 1–8.

Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K., & Liu, X. (2019). A novel neural source code representation based on abstract syntax tree. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 783–794.

Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2020). Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, *48*(1), 1–36.

Zhang, Y., & Li, B. (2020). Malicious code detection based on code semantic features. *IEEE Access*, *8*, 176728–176737.

Zheng, W. [Wei], Gao, J., Wu, X., Liu, F., Xun, Y., Liu, G., & Chen, X. (2020). The impact factors on the performance of machine learning-based vulnerability detection: A comparative study. *Journal of Systems and Software*, *168*, 110659.

Zheng, W. [Wenhao], Zhou, H., Li, M., & Wu, J. (2019). CodeAttention: translating source code to comments by exploiting the code constructs. *Frontiers of Computer Science*, *13*, 565–578.

# KAYNAKLAR DİZİNİ

Zhong, C., Yang, M., & Sun, J. (2019). Javascript code suggestion based on deep learning. *Proceedings of the 2019 3rd International Conference on Innovation in Artificial Intelligence*, 145–149.