FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

ÖN EĞİTİMLİ DİL MODELLERİNİN KOKAN KOD SINIFLAMA PERFORMANSININ ÜÇLÜ KAYIP YÖNTEMİYLE İYİLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

ERTUĞRUL İSLAMOĞLU

İSTANBUL, 2024

FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

ÖN EĞİTİMLİ DİL MODELLERİNİN KOKAN KOD SINIFLAMA PERFORMANSININ ÜÇLÜ KAYIP YÖNTEMİYLE İYİLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

ERTUĞRUL İSLAMOĞLU
(220221002)

Danışman
(Dr. Öğr. Üyesi Ali Nizam)

TÜBİTAK tarafından 123E020 proje numarası ile desteklenmiştir.

İSTANBUL, 2024

FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ
TEZ ONAY FORMU

26/06/2024

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ MÜDÜRLÜĞÜNE

Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Tezli Yüksek Lisans programı öğrencisi 220221002 numaralı Ertuğrul İSLAMOĞLU'nun hazırladığı "Derin Öğrenme Tabanlı Kötü Kod Tespit Sistemi Geliştirilmesi" konulu Yüksek Lisans tezi ile ilgili Tez Savunma Sınavı, 26/06/2024 Çarşamba günü saat 11:30'da yapılmış, sorulara alınan cevaplar sonunda adayın tezinin **Kabulüne [redacted]/Oy Birliği** ile karar verilmiştir.

Tez adı değişikliği yapılması halinde: Tez adının Ön Eğitimli Dil Modellerinin Kokan Kod Sınıflama Performansının Uçta Kayıp Yöntemiyle İyileştirilm şeklinde değiştirilmesi uygundur.

	Jüri Üyesi	Karar
(Danışman)	Dr. Öğr. Üyesi Ali Nizam	Kabul
1.		
	Doç.Dr. Bulut Doğan	Kabul
2.		
	Dr. Öğr. Üyesi Zeki KUŞ	KABUL
3.		
	.....	.....
4.		
	.....	.....
5.		
	(İkinci Danışman)*.....	.....
6.		

**2. Danışman varsa doldurulması gerekmektedir.*

ETİK BİLDİRİM

Bu tezin yazılmasında bilimsel ahlak kurallarına uyulduğunu, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, tezin herhangi bir kısmının bağlı olduğum üniversite veya bir başka üniversitedeki başka bir çalışma olarak sunulmadığını beyan ederim.

Ertuğrul İslamoğlu

TEŞEKKÜR

Tez süreci boyunca bana her türlü desteği veren aileme teşekkür ederim. Çalışmamın bilimsel kalitesini kontrol eden ve beni yönlendiren danışman hocam Dr. Ali Nizam'a ve her türlü soruma ilgiyle cevap veren çalışma arkadaşlarıma da teşekkür ederim.

Ertuğrul İslamoğlu

ÖN EĞİTİMLİ DİL MODELLERİNİN KOKAN KOD SINIFLAMA PERFORMANSININ ÜÇLÜ KAYIP YÖNTEMİYLE İYİLEŞTİRİLMESİ

Ertuğrul İslamoğlu

ÖZET

Tez çalışmasının amacı, geliştiricilerin kodda yaptıkları değişiklikleri ve kötü kokan kod veya kısaca kötü kod (code smell) tespit araçlarının çıktılarını derin öğrenme sistemleriyle analiz ederek bir kötü kod tespit sistemi oluşturmaktır. Analizin derin öğrenme teknikleriyle yapılmasıyla kötü kod tespitinde semantik anlamın dikkate alınması, doğruluk ve performansın arttırılması hedeflenmektedir. Bu hedefe yönelik olarak derin öğrenme alanında kullanılan ön eğitimli dil modellerin, üçlü kayıp tekniğiyle iyileştirilerek kokan kod sınıflama performansının arttırılmasına yönelik bir çalışma yapılmıştır.

Günümüzde büyüyen ve çeşitlenen kod miktarı, kod analizi işlemlerinde ve kod yönetiminde zorluklar meydana getirmiştir. Bununla birlikte İnternetteki büyük miktarda açık kaynak kod içeren GitHub gibi depolar kod analizinde önemli fırsatlar sunmaktadır.

Kod yerleştirmesi (code embedding), kodun semantik anlamını vektörel biçimde saklar. Mevcut kod yerleştirmesi yöntemleri, kaynak kod analizinde çeşitli yazılım mühendisliği görevleri için başarıyla kullanılmış olsa da statik kod analiz araçlarının performansını ve işlevselliğini elde edebilmek için ilave çalışmalara ihtiyaç vardır. Ayrıca, kod yerleştirme modelleri kullanan sistemlerde performansın arttırılması için, görüntü işleme alanında olduğu gibi, yerleştirmeleri iyileştirme için kullanılan ön işleme yöntemlerinin standartlaştırılmasına ihtiyaç vardır.

Bu çalışma, toplanan kötü kodları sınıflandırmak için bir modelin geliştirilmesini ve bu modelin performansını iyileştirmek amacıyla karşılaştırmalı (contrastive) öğrenmenin kod yerleştirmelerine uygulanmasını kapsamaktadır.

Kod kokusu tespiti gibi kod sınıflandırma görevleri için sınıf içi benzerliği güçlendirmek ve farklı sınıflar arasındaki mesafeyi artırmak amacıyla üçlü kayıp tabanlı bir ağ kullanılmaktadır. GitHub'daki açık kaynaklı proje depolarından kod toplanarak deneysel bir veri kümesi oluşturulmuştur.

Çalışmada, yaygın olarak kullanılan, önceden eğitilmiş; BERT, CodeBERT ve GraphCodeBERT dil modellerinin ürettiği kod yerleştirmeleri ve karşılaştırmalı öğrenme ile iyileştirilmiş ve bu iyileştirmenin sınıflamaya etkisi değerlendirilmiştir. Bulgular, ön eğitimli modeller ile oluşturulan yerleştirmelerinin doğrudan kullanımı ile %80-89 arasında bir doğruluk oranı elde edildiğini göstermiştir. Bu doğruluk oranı karşılaştırmalı öğrenme kullanımı ile %7-19 arasında iyileştirilmiştir. Bu sonuçlar, karşılaştırmalı öğrenmenin bir ön işleme adımı olarak önceden eğitilmiş kod yerleştirmeler yaklaşımları için avantajlar sunabileceğini göstermektedir. Sonuç olarak, karşılaştırmalı öğrenme tekniklerinin kod yerleştirme vektörü oluşturma sürecine dahil edilmesi, kod analizinde performans iyileştirmesi için fırsatlar sağlayabilir.

Anahtar Kelimeler: Derin Öğrenme, Kötü Kod, Üçlü Kayıp, Karşılaştırmalı Öğrenme

OPTIMIZING THE CODE SMELL CLASSIFICATION PERFORMANCE OF PRETRAINED LANGUAGE MODELS USING THE TRIPLE LOSS METHOD

Ertuğrul İslamoğlu

ABSTRACT

The aim of this thesis is to create a bad code or smelly code detection system by analyzing the code changes made by developers and the outputs of code smell detection tools with deep learning systems. By using deep learning techniques, it is aimed to take semantic meaning into account in bad code detection and to increase accuracy and performance. Towards this goal, a study has been conducted to improve the performance of the pre-trained language models used in deep learning by using the triple loss technique to improve the performance of bad code classification.

Today, the growing and diversifying amount of code has created difficulties in code analysis and code management. However, repositories such as GitHub, which contain large amounts of open source code on the Internet, offer significant opportunities in code analysis.

Code embedding stores the semantic meaning of code in vector form. While existing code embedding methods have been successfully used in source code analysis for various software engineering tasks, additional work is needed to achieve the performance and functionality of static code analysis tools. Furthermore, to improve the performance of systems using code embedding models, there is a need to standardize the preprocessing methods used to refine embeddings, as in the field of image processing.

This thesis presents the development of a model for classifying collected smelly codes and the application of contrastive learning to code embeddings to improve the performance of this model.

For code classification tasks such as code odor detection, a triple loss-based network is used to strengthen the intra-class similarity and increase the distance between different classes. An experimental dataset was created by collecting code from open-source project repositories on GitHub.

In the study, the widely used, pre-trained BERT, CodeBERT and GraphCodeBERT language models are improved with code embeddings and benchmark learning, and the effect of this improvement on classification is evaluated. The results show that the direct use of the embeddings generated by the pre-trained models yields an accuracy rate between 80-89%. This accuracy was improved by 7-19% with the use of contrastive learning. These results show that contrastive learning can offer advantages for pre-trained code embeddings approaches as a pre-processing step. Consequently, the incorporation of contrastive learning techniques into the code placement vector generation process can provide opportunities for performance improvement in code analysis.

Keywords: Deep Learning, Code Smells, Triplet Loss, Contrastive Learning

ÖN SÖZ

Bu tez, yazılım geliştirme sürecinde önemli bir yer tutan kötü kodların tespit edilmesi ve bu sürecin iyileştirilmesi amacıyla derin öğrenme tekniklerini kullanan bir çözüm sunmayı amaçlamaktadır. Kod kokuları veya kötü kod, yazılımın zamanla bakımını ve genişletilmesini zorlaştıran, ancak doğrudan hataya yol açmayan yapısal kusurlardır. Bu tez, yazılım mühendisliği alanındaki bu önemli soruna yenilikçi bir yaklaşım getirmeyi hedeflemektedir.

Tez çalışması boyunca, derin öğrenme modellerinin kod kokusu tespitinde nasıl kullanılabileceği üzerine yoğunlaşılmış ve bu alanda yapılan literatür taramaları sonucunda mevcut yöntemler incelenmiştir. Ayrıca, önerilen yöntemlerin etkinliğini değerlendirmek üzere kapsamlı deneyler yapılmış ve elde edilen sonuçlar detaylı bir şekilde analiz edilmiştir. Bu çalışmanın yazılım mühendisliği teorik ve pratik uygulamalarına önemli katkılar sağlayacağı düşünülmektedir.

Bu çalışmanın gerçekleştirilmesinde emeği geçen ve desteklerini esirgemeyen birçok kişi bulunmaktadır. Öncelikle, tez danışmanım Dr. Öğr. Üyesi Ali NİZAM hocama, değerli bilgi ve yönlendirmeleri için sonsuz teşekkürlerimi sunarım. Ayrıca, aileme ve arkadaşlarıma, bu zorlu süreçte verdikleri manevi destek ve sabırları için minnettarım. Bu tezin her aşamasında yanımda olan, fikirleriyle beni motive eden ve çalışmalarımın gelişmesine katkıda bulunan tüm proje arkadaşlarıma teşekkür ederim.

Bu çalışma, Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) tarafından “Derin Öğrenme ve Kod Tarihçesi Tabanlı Yazılım Kod Kalite Analiz Sistemi Geliştirilmesi” isimli, 123E020 numaralı proje kapsamında desteklenmiştir. Destekleri için TÜBİTAK'a da teşekkür ederim.

Haziran, 2024 Ertuğrul İslamoğlu

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

KISALTMALAR

bkz.	Bakınız
DNN	Derin Sinir Ağı
t-SNE	T-dağıtılmış stokastik komşu yerleştirme
BERT	Dönüştürücülerden Çift Yönlü Kodlayıcı Yerleştirmeleri
LLM	Büyük Dil Modeli
AST	Soyut Sözdizimsel Ağaç
LN	Katman Normalizasyon
BS	Yığın boyutu
LR	Öğrenme Oranı

GİRİŞ

Kod kalite standartlarının yükseltilmesi, yazılım mühendisliğinde çok önemli bir konudur. Bu konuda ilk olarak Cunningham tarafından (Cunningham, 1992) *Teknik borç* kavramı ortaya atışmıştır. Teknik borç "*tam doğru olmayan ama düzeltmeyi ertelediğimiz kod*" olarak tanımlanmıştır. Teknik borcun yazılım kalitesindeki etkileri ampirik olarak kanıtlanmış olsa da, teknik borcun nerede, ne zaman ve nasıl ortaya çıktığı üzerine hala yeterince ampirik kanıt yoktur (Tufano vd., 2015).

Nesne tabanlı bağlamda teknik borç yaratacak kodların tespitinde yardımcı olması için Fowler ve Beck, 22 yazılım yapısını, kötü kokan kod (code smells) ismiyle öne sürmüşlerdir (Martin Fowler vd., 1999). Kötü kokan kod (kısaca "kokan kod" veya "kokular"), yani kötü tasarım ve uygulama seçimlerinin belirtileri, teknik borca katkıda bulunan ve bir yazılım sisteminin sürdürülebilirliğini olumsuz etkileyen önemli bir faktörü temsil eder (Tufano vd., 2015).

Kod kokuları koddaki hatalar değildir, yani yazılımın çalışmasını engellemezler. Kokular, bir yazılım sisteminin tasarım kısmındaki zayıflıklardır ve geliştirmeyi yavaşlatabilir ya da gelecekte başarısızlık veya hata tehdidini artırabilir (Brown & Greer, 2023).

Kod kokularını tespit etmek için farklı yöntemler olmakla birlikte (Rasool & Arshad, 2015), bu tez çalışmasında, DNN (Deep Neural Network – Derin Sinir Ağı) ağı kullanılarak derin öğrenme tabanlı bir yaklaşımda bulunulmuştur. Ağda test edilmek üzere gerekli verilerin çıkarılması için SonarQube aracı kullanılmıştır.

Kod verilerini bir derin ağa besleyebilmek için bu verilerin sayısallaştırılması gereklidir. Doğal dil işlemede saf verinin sayısal bir formata dönüştürülmüş haline yerleştirme (embedding) denir. Kod yerleştirmeleri, kaynak kod analizi için kodu, dağıtılmış vektör yerleştirmelerine dönüştürür (Sui vd., 2020) ve kod özetleme ve anlamsal etiketleme, kod klon tespiti, hata raporu, yazılım kalite değerlendirmesi, kod kokusu tespiti, kod incelemesi vb. gibi çeşitli yazılım mühendisliği görevlerinde kullanılır (Hou vd., 2023), (Büyük & Nizam, 2023).

Etkili yerleştirmenin keşfedilmesi önemli ve yeni bir araştırma alanıdır. Yazılım mühendisliği araştırmacıları; AST'deki (Abstract Syntax Tree – soyut sözdizimsel ağaç) yollar gibi sözdizimsel bir ağaç yerleştirmesi kullanan (Alon vd., 2019), (Yahav & Levy, 2019), veya BERT (Bidirectional Encoder Representations From Transformers, Dönüştürücülerden Çift Yönlü Kodlayıcı Yerleştirmeleri) (Devlin vd., 2019) gibi LLM (large language model, büyük dil modeli) kullanarak kodu doğrudan işleyen, veya CodeBERT (Feng vd., 2020), GraphCodeBERT (Guo vd., 2020) gibi kod semantiğini anlamak için özel eğitilmiş LLM kullanan veya derleyici ara yerleştirmeleri (compiler intermediate representation) gibi düşük seviye kod yapılarını kullanan (Ben-Nun vd., 2018), farklı kod yerleştirme teknikleri geliştirilmiştir.

Yerleştirme performansını artırmak için çeşitli optimizasyon teknikleri kullanılmaktadır. İnce ayar (fine-tuning), önceden eğitilmiş modelleri belirli görevlere uyarlamak ve böylece bir dizi doğal dil işleme uygulamasında performanslarını artırmak için LLM çalışmalarında baskın optimizasyon yöntemi haline gelmiştir. Hiper parametre optimizasyonu, belirli görevlerde optimum performans elde etmek için dil modellerinin hiper parametrelerine ince ayar yapmanın önemini vurgulayan bir diğer önemli yaklaşımdır (Hou vd., 2023). Buna ek olarak, daha iyi yerleştirmeler öğrenmek için veri artırma (data augmentation) teknikleri önerilmektedir (Li vd., 2022). Ancak, kaynak kod modelleme ve yerleştirme optimizasyonunda tam olarak araştırılmamıştır (Zhuo vd., 2023).

Bu çalışma, üçlü kayıp (triplet loss) tabanlı bir derin sinir ağı (DNN) ile yerleştirme sınıfları arasındaki mesafeyi düzenleyerek kod yerleştirmelerini iyileştirmeyi amaçlamaktadır. Bilgisayar görüşünde; yüz tanıma, görüntü erişimi, kişi tespiti gibi alanlarda kullanılan üçlü kayıp (Dong & Shen, 2018), özünde aynı sınıfa sahip olan örneklerin yerleştirmelerinin birbirlerine yaklaştırılması ve farklı olan sınıfların uzaklaştırılması üzerinedir. Bu kayıp yöntemi elde edilen yerleştirmeler üzerinde çalıştırılmış ve başarıda ciddi bir artış gözlemlenmiştir.

Tez kapsamında yapılan temel katkılar: (1) Kod kokusu tespiti için önceden eğitilmiş yerleştirme modelleri için hiper parametre optimizasyonu uygulanmış, kaynak-hedef çifti veri kümemizle ince ayar yapılmıştır, (2) Kod kokusu tespitinin doğruluğunu artırmak için üçlü kayıp kullanılarak ön eğitimli yerleştirmeler optimize

edilmiştir. Önerilen üçlü kayıp yöntemlerinin avantajları teorik olarak analiz edilmiş ve test bulgularıyla gösterilmiştir.

BİRİNCİ BÖLÜM

1. LİTERATÜR ARAŞTIRMASI VE TEORİK ÇERÇEVE

Bu bölümde tez çalışmasının konusu ile alakalı olan diğer çalışmalardan kısaca bahsedilecektir. Üzerinde durulacak alanlar sırasıyla kod kokularının tespiti, kod yerleştirmeleri ve kod yerleştirmelerinin iyileştirilmesidir.

1.1. KOD KOKUSU

Yazılım projelerinde kod kalite problemlerinin nasıl, ne zaman ve neden ortaya çıktığı tam tespit edilemektedir (Tufano vd., 2015). Bu durum, kodddaki kalite sorunlarıyla etkili ve verimli bir şekilde yönetilmesinin önünde bir engel teşkil etmektedir. Fowler ve Beck, nesne yönelimli bağlamda sorunlu kodun belirlenmesine yardımcı olmak amacıyla, "kötü kokular" olarak adlandırdıkları 22 yazılım yapısını kötü kodun göstergeleri olarak tanıtmışlardır (Martin Fowler vd., 1999). Bu kötü kokuların, yazılım geliştiricilere yazılımın ne zaman yeniden düzenlenmesi (refactor) gerektiğine karar vermelerinde yardımcı olmaları amaçlanmıştır (Mäntylä vd., 2003).

Geçmişte, kod kokularının geliştiriciler için ne kadar önemli olduğu, kod kokularının bir yazılım sisteminde ne kadar uzun süre kalma eğiliminde olduğu ve kod kokularının değişiklik ve hata eğilimlerinde artış veya yazılımın anlaşılabilirliğinde ve sürdürülebilirliğinde azalma gibi yan etkileri çeşitli çalışmalarla araştırılmıştır (Tufano vd., 2015).

Fowler tarafından tanımlanan *uzun metot, farklı arayüzlere sahip alternatif sınıflar* ve *mesaj* (Fowler, 2018) gibi 22 kod kokusu yanında *ölü kod* gibi Fowler tarafından başlangıçta tanımlanmamış bazı ek kod kokuları da ortaya çıkmıştır. Bazı kod kokuları (*özellik kıskançlığı* gibi) bireysel metodlarda, bazıları (*tembel sınıf* gibi) bireysel sınıflarda ve bazıları (*uygunsuz samimiyet* gibi) sınıflar arasındaki ilişkilerde mevcuttur. Mantyla vd., (Mäntylä vd., 2003) Fowler tarafından tanımlanan 22 kod kokusunu ve bir ek kod kokusunu yedi kategoriye ayırmıştır: gereksiz kod şişiriciler (bloaters), nesne yönelimini kötüye kullananlar, değişim önleyiciler, vazgeçilebilirler, kapsülleyiciler, bağlayıcılar ve diğerleri. Örneğin şişiriciler, çalışılması özellikle zor olacak kadar artan yöntem ve sınıfları ifade eder; genellikle

hemen ortaya çıkmak yerine program ve kod tabanı geliştikçe zaman içinde birikirler. Nesne yönelimini kötü kullananlar, nesne yönelimli programlama ilkelerinin eksik veya yanlış uygulamalarını ifade eder. Diğer kategorilerin açıklamaları Mantyla vd.'nin (Mäntylä vd., 2003) taksonomisinde bulunabilir.

1.2. KOD KOKULARININ TESPİTİ

Yazılım mühendisliği literatüründe farklı kod kokusu türleri mevcuttur: yinelenen kod, uzun yöntem, büyük sınıf, uzun parametre listesi vb. (Martin Fowler vd., 1999). Bu kokular bir kodu yeniden düzenlemenin gerektiğini belirten sinyaller olarak düşünülebilir (Alazba & Aljamaan, 2021). Bu nedenle, yeniden düzenlemek için kod kokularını tespit etmek oldukça önemlidir.

Alazba vd.'ye göre kod kokusu tespiti için üç ana yaklaşım vardır: metrik tabanlı, kural tabanlı ve makine öğrenimi tabanlı (Alazba & Aljamaan, 2021). Metrik tabanlı yaklaşım, kalıtım, boyut ve uyum gibi kalite metriklerinin tanımlanmasını ve ardından her metrik için bir eşik değeri belirlenmesini gerektirir. Ancak, bu yaklaşımda doğru eşik değerini seçmek ve tanımlamak basit bir görev değildir. Daha sonra, kural tabanlı yaklaşımda, alan uzmanlarının her bir kod kokusunu tanımlamak için belirli kuralları belirlemeleri gerekir. Bu kurallar bazen manuel olarak oluşturulur ve alana özgü bir dil kullanılır. Bu yaklaşımlarda yazılım mühendislerinden istenen çaba ve bilişsel yük nedeniyle, son araştırmalar daha çok makine öğrenimi yaklaşımlarına yönelmiştir (Alazba & Aljamaan, 2021).

Son zamanlarda, makine öğrenimi sınıflandırıcılarının kod kokusu tespit kuralları ve eşikleri oluşturduğu yazılım kod kokusu tespitinde makine öğrenimi sınıflandırıcılarının uygulanması araştırılmıştır. Bu yaklaşımda, yazılım metrikleri kaynak koddan çıkarılır ve otomatik olarak algılama kuralları ve eşikleri üretmek için makine öğrenimi sınıflandırıcılarına beslenir. Bu sayede yazılım mühendislerinin harcadığı efor azaltılmakta ve tespit süreci kolaylaştırılmaktadır. Kod kokularının tespit edilmesinde çeşitli makine öğrenimi sınıflandırıcıları kullanılmıştır: karar ağaçları, destek vektör makineleri, rastgele orman ve naif Bayes. Ancak, farklı kod kokuları üzerinde etkili bir algılama performansına sahip bir sınıflandırıcı bulmak henüz başarılamamıştır (Alazba & Aljamaan, 2021).

1.3. DERİN ÖĞRENME

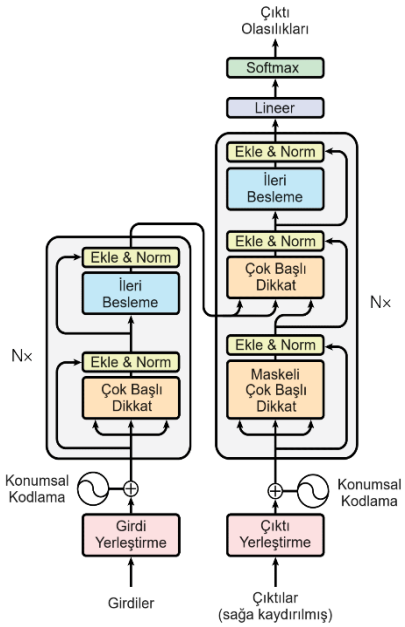
Son zamanlarda, makine öğrenimi (machine learning), araştırmalarda çok yaygın hale gelmiştir ve metin madenciliği, spam algılama, video önerme, görüntü sınıflandırma ve multimedya kavram erişimi gibi çeşitli uygulamalara dahil edilmiştir. Farklı makine öğrenimi algoritmaları arasında derin öğrenme (deep learning) bu uygulamalarda çok yaygın olarak kullanılmaktadır. Derin ve dağıtık öğrenme alanlarında sürekli yeni çalışmaların ortaya çıkması, hem veri elde etme yeteneğindeki öngörülemeyen büyümeden, hem de yüksek performans hesaplama (high performance computing) gibi donanım teknolojilerinde kaydedilen şaşırtıcı ilerlemeden kaynaklanmaktadır (Alzubaidi vd., 2021).

Derin öğrenme, geleneksel sinir ağından (neural network) türetilmiştir ancak öncüllerinden önemli ölçüde daha iyi performans gösterir. En son geliştirilen derin öğrenme teknikleri, ses ve konuşma işleme, görsel veri işleme, doğal dil işleme dahil olmak üzere çeşitli uygulamalarda önceki modellere göre yüksek performans elde etmiştir.

1.3.1. Dönüştürücü

Dönüştürücü (transformer) (Vaswani vd., 2017) doğal dil işleme, bilgisayarla görme ve konuşma işleme gibi çeşitli alanlarda yaygın olarak benimsenen önemli bir derin öğrenme modelidir.

Diziden diziye (sequence-to-sequence) bir model olan dönüştürücü, her biri L özdeş bloklardan oluşan bir yığın olan bir kodlayıcı ve bir kod çözücüden oluşur. Her bir kodlayıcı bloğu temel olarak çok başlı bir öz dikkat (multi-head self-attention) modülünden ve konum bazlı bir ileri besleme ağından (feed-forward network) oluşur. Daha derin bir model oluşturmak için, her modülün etrafında bir artık bağlantı (residual connection) ve ardından katman normalleştirme (layer normalization) modülü kullanılır. Kodlayıcı bloklarla karşılaştırıldığında, kod çözücü bloklara ek olarak çok başlı öz dikkat modülleri ile konum bazlı ileri besleme ağları arasına çapraz dikkat modülleri ekler. Ayrıca, kod çözücüdeki öz dikkat modülleri, her bir konumun sonraki konumlara katılmasını önleyecek şekilde uyarlanmıştır. Dönüştürücünün genel mimarisi Şekil 1.1'de gösterilmektedir.



Şekil 1.1 Dönüştürücü mimarisi (Vaswani vd., 2017)

Sonuç olarak dönüştürücü, özellikle önceden eğitilmiş modeller, doğal dil işlemede tercih edilen bir mimari haline gelmiştir (T. Lin vd., 2021).

1.4. KOD YERLEŞTİRME ÜRETEN TEKNİKLER

Makine öğrenimi ve derin öğrenme tekniklerinin kod analizine uygulanması nispeten yeni bir araştırma alanıdır. İlk zamanlarda, Word2Vec (Mikolov vd., 2013) tabanlı kod yerleştirme teknikleri, kaynak koda bir dizi metinsel belirteç (token) olarak yaklaşmıştır. Word2Vec, sözlüğü (vocabulary) bir Huffman ikili ağacı olarak düzenler ve değerlendirilen çıktı birimlerinin sayısını ve karmaşıklığı azaltmak için daha sık kullanılan kelimelere daha kısa ikili kodlar atar (Mikolov vd., 2013). Word2Vec yüksek kaliteli kelime vektörleri oluşturabilir ancak yazılım kaynak kodu doğal dilden farklıdır (Sui vd., 2020). Kod, çalıştırılabilirdir ve resmi sözdizimi ve semantiğe sahiptir (Allamanis vd., 2017).

Bazı kod yerleştirmeleri teknikleri, çizge bilgisini korurken bir çizgeyi düşük boyutlu bir uzaya dönüştüren çizge yerleştirmeye dayanmaktadır. AST'deki farklı anlamsal birimler, kaynak kodun karmaşık yapısını temsil eder. Kod bileşenlerini bir AST yolları koleksiyonuna ayrıştırdıktan sonra sinir ağı, her yolun atomik temsilini ve yolların aralarındaki birleşme modellerini öğrenir (Alon, Zilberstein, vd., 2018). Code2Vec (Alon, Zilberstein, vd., 2018) PathPair2Vec (Shi vd., 2020) ve Code2seq

(Alon, Brody, vd., 2018) gibi birçok kod temsil modeli vardır. Code2Vec, anlamsal özellikleri tahmin etmek için bir kod verisini sabit uzunlukta tek bir kod vektörüne dönüştürür(Alon, Zilberstein, vd., 2018).

1.4.1. Kod Yerleştirme Üreten Ön Eğitimli Dil Modelleri

Ön eğitim, daha büyük ve daha iyi bir modelin eğitilmesini sağlayarak yardımcı olan başka bir yöntemdir. İlk zamanların doğal dil tabanlı ön eğitimli modellerinde, sözdizimsel ve anlamsal yapılar gibi koda özgü özellikler uygun şekilde dikkate alınmayabilir ve performansları yeterli olmayabilir (Niu vd., 2023). Son zamanlarda, büyük ölçekli ön eğitim tekniklerinin ortaya çıkmasıyla birlikte, kaynak kod için çeşitli ön eğitim görevlerine sahip bazı büyük modeller önerilmiş ve önceki modellerden önemli ölçüde daha iyi performans göstermişlerdir (Li vd., 2022). Kod yerleştirme tabanlı derin öğrenme çözümleri için yinelemeli (recurrent) sinir ağı (Hochreiter & Schmidhuber, 1997) ve dönüştürücü (Vaswani vd., 2017) tabanlı sistemler kullanılmaktadır. Dönüştürücü yapısının avantajları, bu alandaki araştırmaların yoğunlaşmasına yol açmıştır. CodeBERT (Feng vd., 2020) ve GraphCodeBERT (Guo vd., 2020) gibi birçok ön eğitimli kod temsil modelleri kod analizinde başarıyla kullanılmaktadır. Ayrıca, yakın zamanda geliştirilen BERT (Devlin vd., 2019) gibi dönüştürücü tabanlı metin tabanlı yöntemlerin de kod temsilinde başarılı sonuçlar verdiği gösterilmiştir.

1.5. YERLEŞTİRME İYİLEŞTİRME TEKNİKLERİ

Kod yerleştirme performansını artırmak için ince ayar, ağ ağırlıklarını düzenleyerek önceden eğitilmiş modelleri belirli görevlere uyarlar. Evrensel kod özelliklerini çıkarmak için temsil üzerinde çizge sinir ağlarını ön eğitime tabi tuttuktan sonra, çeşitli aşağı akış uygulamalarını desteklemek için ince ayar yapma olasılığı vardır (Liu vd., 2021). İnce ayar stratejileri ile transfer öğrenme ve çoklu görev öğrenme, tüm görevlerde tek görev tabanlı modellerden daha iyi performans gösterir Yerleştirme optimizasyonu için özellik çıkarmaya dayalı farklı yöntemler de kullanılır. Konumsal kodlama (positional encoding) olarak kullanmadan önce k en düşük özdeğere sahip özvektörleri ön işleme tabi tutmak, yapıya duyarlı bir dönüştürücü elde etmek için kullanılan bir diğer yöntemdir (Geisler vd., 2023). Bazı

çalışmalar, programların dinamiğini keşfetmek ve tamamlayıcılar (Huang vd., 2023) veya kod geçmişi olarak kodun özellik temsillerine yerleştirmek için program test durumları gibi ek kaynaklardan ortaya çıkan bilgileri önermektedir. Son zamanlarda, kod analizi görevleri için kontrastlı öğrenme kullanılmıştır (Bui vd., 2021).

1.5.1. Karşılaştırmalı Öğrenme

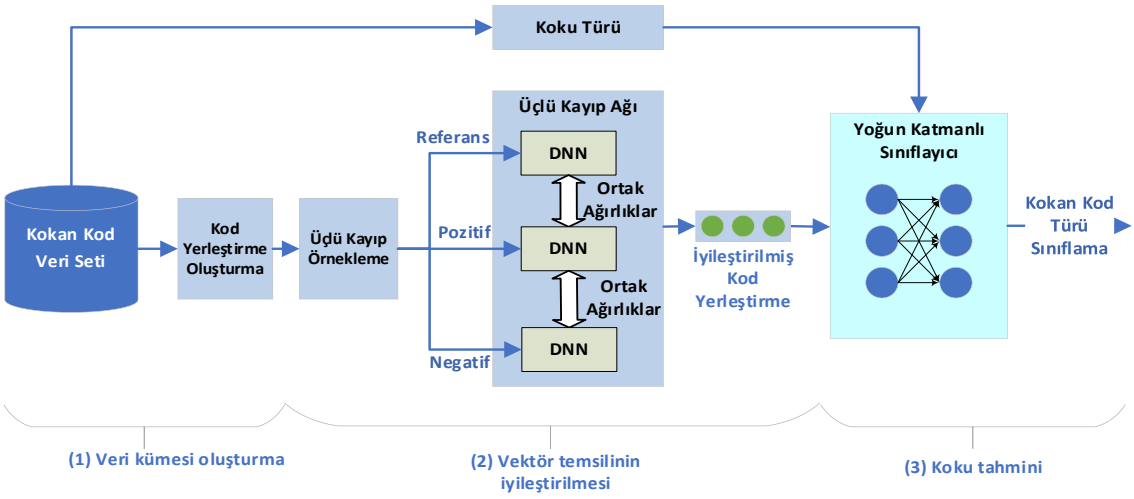
Karşılaştırmalı öğrenme (contrastive learning), pozitif ve negatif örnek çiftlerini karşılaştırarak anlamlı temsiller çıkarmaya odaklanır. Ana prensibi, öğrenilen bir yerleştirme uzayında benzer örnekleri birbirine daha yakın ve benzer olmayan örnekleri daha uzak konumlandırmaktır. Yöntem, gizli uzayda (latent space) bir zıtlık kaybı kullanarak aynı örneğin farklı şekilde artırılmış görünümleri arasındaki uyumu en üst düzeye çıkarır (Zhang vd., 2024). Kontrastlı eğitimin, yüz algılamanın görsel temsilleri gibi alanlarda kullanımı vardır (Chen vd., 2020). FaceNet, kontrastlı eğitimin ilk uygulamalarından birisidir (Schroff vd., 2015). Yüz tanıma performansını artırmak amacıyla yerleştirme işlemini optimize etmek için yeni bir çevrimiçi üçlü madencilik yöntemi kullanılarak oluşturulan kabaca hizalanmış eşleşen / eşleşmeyen yüz eşleşmelerinin üçlülerini kullanır (Schroff vd., 2015).

Görüntü işleme alanından uyarlanan üçlü kayıp (triplet loss), kod analizi için kod klon tespiti ve veri arttırma görevlerinde yaygın olarak kullanılmaktadır. Çeşitli çalışmalar, denetimli ve denetimsiz yöntemler kullanarak kod analizinde karşılaştırmalı öğrenmeyi kullanmıştır. Denetimli karşılaştırmalı öğrenme, etiketleri ile birlikte veri noktaları çiftleri üzerinde eğitilmiş bir model kullanarak, benzer ve benzer olmayan örnekler arasında ayrım yapmak üzere modelleri açıkça eğitmek için etiketli verileri kullanır. Denetimsiz eğitim için, Bui vd., etiketli verileri kullanmadan öğrenmek için, semantiğini değiştirmeden bir kodun birden çok sürümünü oluşturan kendi kendini denetleyen (self-supervised) bir karşılaştırmalı öğrenme çerçevesi önermiştir (Bui vd., 2021).

İKİNCİ BÖLÜM

2. YÖNTEM

Kullanılan ön eğitim teknikleri ve uyarlanma yöntemi Şekil 2.1'de açıklanmıştır. Önerilen sistemin yaygın olarak kullanılan BERT, CodeBERT ve GraphCodeBERT dil modelleri ile karşılaştırmalı bir değerlendirmesi yapılmıştır. Bu bölümde bu dil modelleri tanıtılmış ve kullanılacak olan karşılaştırmalı öğrenim yöntemi olan üçlü kayıp (triplet loss) tanımlanmıştır.



Şekil 2.1 Üçlü ağ kod sınıflama modeli

Üçlü kayıp ile ön eğitimli modellerin kod sınıflama performansı artışını denemek için BERT, CodeBERT ve GraphCodeBERT modelleri kullanılmıştır. Metin ve kod tabanlı ön eğitim yaklaşımlarının temel bir karşılaştırmasını yapabilmek için metin tabanlı BERT modeli incelenmiştir. CodeBERT, BERT yapısını bir dönüştürücü mimarisi ve kod ile yorum satırları arasındaki ayrımla genişletmektedir. Böylece kod ve metnin birlikte kullandığı bir model de test edilmiştir. GraphcodeBERT, kontrol akış bilgisini de kullanarak kodun doğal yapısının araştırılmasını sağlar. Böylece, metin tabanlı, metin + kod tabanlı ve koda

özgü optimizasyon içeren üç model üzerinden genel bir iyileştirme yöntemi sağlanması hedeflenmiştir. Bu modeller 512 belirteçle sınırlıdır. İncelenen kodların uzun olması sebebiyle sisteme verilmeden önce yorum satırları kaldırılarak sistemin sadece kod analizine yoğunlaşması hedeflenmiştir.

2.1. STATİK KOD ANALİZ ARACI İLE KOKAN KOD VERİ SETİ OLUŞTURMA

SonarQube, 25'ten fazla programlama dilinde hataları, kod kokularını ve güvenlik açıklarını tespit etmek için kodun statik analizi ile otomatik incelemeler gerçekleştiren açık kaynaklı bir platformdur. SonarCloud ise, SonarQube kullanılarak incelenmiş kod depolarının (repository) ve depolarda tespit edilen hata, koku ve güvenlik açıklarının saklandığı halka açık bir platformdur.

SonarCloud üzerinden elde edilen veriler ile oluşturulmuş veri setleri çalışmada kullanılmıştır. Kokan kod içeren metod örnekleri SonarCloud'dan toplanmıştır. Kod örnekleri, DNN ağı için girdi, bu örneklerin sahip oldukları kod kokuları ise temel doğru (ground truth) olarak kabul edilerek ağ ile koku tespiti için sınıflandırma (classification) yapılmıştır. Toplanan 5 farklı kokan koda sahip olan örnekler ile çok sınıflı sınıflandırma (multiclass classification) yapılmıştır.

Kokan kaynak kodlarıyla incelenecek veri setlerinin oluşturulması amacıyla SonarCloud'un Rest API'sinden yararlanıldı. Bu API, üzerinde analiz yapılmış farklı kod depolarındaki kokan kod örneklerini verilen parametre ve filtreler doğrultusunda bularak json formatında yerel ortamınıza aktarmanızı sağlamıştır. "https://sonarcloud.io/api/issues/search" adresine aşağıdaki parametreler ile atılan istekler doğrultusunda veri seti oluşturuldu; bkz. Çizelge 2.1.

Çizelge 2.1 REST API yapısı

Anahtar	Değer	Açıklama
resolutions	FIXED	Koddaki kokunun durumunu belirtir, çözülmüş kokuları alarak kodun önceki ve sonraki hali elde eder.
ps	500	Tek bir istek içerisinde dönebilecek koku sayısı belirtir. Maksimum 500.
p	1	Toplamda 2500 adet kokan kod varsa bu parçalara bölünür ve her kokan kod için bir istek atılır.
languages	Java	Programlama dili seçmek için kullanılır.
componentKeys	my-example_my-exaple	Üzerinde arama yapılacak kod deposunun Id tanımıdır.
rules	java:S1172	İstenilen koku türü burada belirtilir. Boş bırakıldığı takdirde tüm koku türlerinden dönüt gelebilir.
additionalFields	_all	Kokunun ayrıntılarıyla alakalı hangi alanların dönmesinin istendiği burada belirtilir. "_all" ile tüm alanlar döndürülür.

Açık kaynak kodlu projelerde kod değişimlerinin toplanması için GitHub API kullanılmıştır. SonarCloud API ile SonarCloud ortamında çok karşılaşılan kötü kod örnekleri toplanmıştır. Çalışmada işlenen kokuların anahtarları ve açıklamaları, örnekler ile Çizelge 2.2'de verilmiştir. SonarQube içerisinde her koku için bir anahtar Id değeri bulunmaktadır. Çalışma kapsamında kokan kod örnekleri, bu Id değerleri ile etiketlenmiştir.

Çizelge 2.2 Çalışmada kullanılan kokan kod bilgileri

Koku Id	Açıklama	İhlal Eden Örnek Kod

java:S100	Yöntem adları bir adlandırma kuralına uygun olmalıdır.	public short getDefaultValueAs_short(){
		try {
		return initExpr == null ? 0 : Short.parseShort(initExpr);
		} catch (NumberFormatException nfe) {
		return StringUtils.isEmpty(initExpr) ? 0 :
		CoreComponentsBuilder.get().getMVELExecutor().eval(initExpr,
		Short.class);
		}
		}

java:S1161	"@Override" metotları geçersiz kılmak ve uygulamak için kullanılmalıdır	public String toString(){
		return symbol;
		}

java:S1452	Jenerik joker değişken türleri dönüş türlerinde kullanılmamalıdır	public static AbstractIteratorAssert<?, T> then(Iterator<? extends T> actual){
		return assertThat(actual);
		}

java:S119	Tür parametre adları bir adlandırma kuralına uygun olmalıdır	protected MH handler(Class<MH> handlerClass){
		return getMetadataHandlerProvider().handler(handlerClass);
		}

java:S1172	Kullanılmayan fonksiyon parametreleri kaldırılmalıdır	private void setPartOfDayFromHour(int hour){
		setAfternoon(true);
		setMorning(true);
		}

Çizelge 2.3'te ise her koku için toplam kod satır sayısı ve kurulacak sistemde atanan kimlik değeri verilmiştir.

Çizelge 2.3 Metot düzeyinde dengeli kokan kod veri kümesi (Her sınıfta 500 örnek)

Koku Id	Sınıf Id değeri	Referans	Toplam Kod Satır Sayısı
java:S100	0	https://rules.sonarsource.com/java/RSPEC-100/	5269
java:S1161	1	https://rules.sonarsource.com/java/RSPEC-1161/	3140
java:S1452	2	https://rules.sonarsource.com/java/RSPEC-1452/	2285
java:S119	3	https://rules.sonarsource.com/java/RSPEC-119/	4172
java:S1172	4	https://rules.sonarsource.com/java/RSPEC-1172/	5960
Toplam			20826

2.2. ÖN EĞİTİMLİ DİL MODELLERİ

Kodun incelenebilmesi için sayısallaştırılması ve kod yerleştirme (embedding) olarak isimlendirilen vektörel bir gösterime dönüştürülmesi gerekir. Farklı önceden eğitilmiş modeller için ayrı ayrı performans testleri gerçekleştirilmiştir. Performans testleri sonuçları aşağıda kod kalite analiz çalışmalarında ayrı ayrı sunulmuştur. Burada uygulanan modellerin genel yapıları ve bunların çalışmada kullanım şekilleri aşağıda kısaca sıralanmıştır.

Çalışmanın bu kısmında üçlü kayıp ile ön eğitimli modellerin kod sınıflama performansı artışını denemek için BERT, CodeBERT ve GraphCodeBERT modelleri kullanılmıştır. Çalışmada, metin tabanlı, metin + kod tabanlı ve koda özgü optimizasyon içeren üç model üzerinden genel bir iyileştirme yöntemi sağlanması hedeflenmiştir. Metin ve kod tabanlı ön eğitim yaklaşımlarının temel bir karşılaştırmasını yapabilmek için metin tabanlı BERT modeli incelenmiştir. CodeBERT, BERT yapısını bir dönüştürücü mimarisi ve kod ile yorum satırları arasındaki ayrımla genişletmektedir. Böylece kod ve metnin birlikte kullandığı bir model de test edilmiştir. GraphCodeBERT, kontrol akış bilgisini de kullanarak kodun doğal yapısının araştırılmasını sağlar. Bu modeller 512 belirteçle sınırlıdır. İncelenen kodların uzun olması sebebiyle sisteme verilmeden önce yorum satırları kaldırılarak sistemin sadece kod analizine yoğunlaşması hedeflenmiştir.

2.2.1. BERT

BERT, tek yönlü standart dil modelleri iyileştirmek için tüm katmanlarda hem sol hem de sağ bağlamı ortaklaşa ele alarak, etiketlenmemiş bir metinden derin çift yönlü yerleştirmelere ön eğitim uygulamak için tasarlanmıştır (Devlin vd., 2019). Ön eğitim ile üretilen yerleştirme vektörleri, alana özel işlemler gerektiren birçok göreve özel mimari tasarlamaya olan ihtiyacı azaltmaktadır. BERT $[C_{[CLS]}, C_1, C_1, \dots, C_{[SEP]}]$ şeklindeki giriş vektör yerleştirmelerini bir dizi transformers katmanından geçirdikten sonra $[T_{[CLS]}, T_1, T_1, \dots, T_{[SEP]}]$ şeklinde bir çıkış vektörüne dönüştürür (Lin vd., 2020). Giriş vektörleri, belirteç (token), segment ve konum yerleştirmelerinin öğe bazında toplamını içerir. BERT attention katmanı her bir belirteç için yerleştirme vektörünü zenginleştirir. Giriş dizisindeki her bir belirteci diğerleriyle karşılaştırarak ilişkisel ve bağlamsal karmaşık ve üst düzey ilişkileri çözümleyebilir. Ön eğitim sırasında model, farklı ön görevler üzerinden etiketlenmemiş verilerle eğitilir. BERT, giriş dizisinden bir belirtecini maskeler ve modelden "tahmin etmesini" ister. Model, tahminlerde bulunmak için maskelenmiş belirtecin sol ve sağ bağlamlarını aynı anda kullanabilir.

BERT modelinin mimarisi 12 dönüştürücü bloğundan oluşan bir kodlayıcı, 12 dikkat katmanı ve 768 gizli durum tanımından oluşur. Tüm modelin giriş ve çıkış uzunlukları 512 olarak ayarlanmıştır. Kendi kendini denetleyen öğrenme aşamasında, belirteçlerin %15'i benzersiz bir maske belirteciyle maskelenir. İnce ayar aşamasında BERT modeli önceden eğitilmiş parametrelerle başlatılır ve tüm parametreler bir cümle çifti (örneğin, $< Soru, Cevap >$) gibi alana özel etiketli veriler kullanılarak iyileştirilir. Cümle kavramı gerçek bir dilsel cümle yerine, bitişik metnin keyfi seçilen bir aralığı olabilir. Pek çok yerleştirme tekniğinde, yalnızca cümle yerleştirmeleri akış görevlerine aktarılır; BERT'te ise sonraki görevlere tüm parametreleri aktarır. BERT, giriş sırası hakkında ek bilgi sağlamak için özel belirteçler kullanır:

Kelime Belirteçleri: Bunlar, giriş metnindeki gerçek kelimeler veya alt kelimelerdir. Kelimeleri daha küçük birimlere ayıran ve bunları alt kelime belirteçlerinin sözlüğü olarak temsil eden WordPiece ile belirteç oluşturulur.

Özel sınıflandırma belirteci [CLS]: Her giriş dizisinin başına eklenir ve "sınıflandırma" belirtecini temsil eder. Sınıflandırma veya regresyon gibi alana özel görevler için tüm dizinin sabit boyutlu bir yerleştirmesini elde etmek için kullanılır.

Ayıraç [SEP]: Bu belirteç girişteki iki diziyi ayırmak için kullanılır. Giriş olarak birden fazla dizi sağlandığında, bir dizinin sonunu ve diğerinin başlangıcını belirten cümle çiftleri arasına eklenir.

Parça Yerleştirmeleri: BERT, giriş dizisindeki farklı bölümler veya cümleleri ayırt etmek için bölüm yerleştirmeleri kullanır. Soru yanıtlama veya doğal dil çıkarımı gibi birden fazla dizi içeren görevler için her diziye benzersiz bir bölüm kimliği (genellikle 0 veya 1) atanır.

Model N adet dönüştürücü katmanı kullanarak kod H^n yerleştirme vektörünü oluşturur. Burada kullanılan formüller:

$$H^n = transformers_n(H^{n-1}), n \in [1, N].$$

$$G^n = LN(MultiAttn(H^{n-1}) + H^{n-1})$$

$$H^n = LN(FFN(G^n) + G^n)$$

G^n çok başlı öz-dikkat katmanın çıktısıdır. FFN iki katmanlı ileri beslemeli bir ağdır. LN katman normalizasyon operasyonunu gösterir. BERT ağının son katmanı ek normalizasyon gerçekleştirmektedir.

Çalışmada H^n çıktı vektörü üretilerek sınıflama aşamalarında kullanılmıştır.

2.2.2. CodeBERT

CodeBERT, doğal dil ile programlama dili arasındaki anlamsal bağlantıyı yakalamak için tasarlanmıştır. Dönüştürücü tabanlı bir ağ mimarisi kullanılarak oluşturulmuş ve değiştirilen belirteç algılama görevini içeren hibrit bir amaç fonksiyonuyla eğitilmiştir. Kod için programlama dili ve yorum satırları için doğal dil belirteçlerini birleştirmek için tasarlanmıştır (Feng vd., 2020). CodeBERT yorumları, kaynak kodunu ve değişkenleri ardışıl giriş X olarak alır ve bu giriş değerlerini giriş vektörü H^0'a çevirir. Kaynak kodu $C = \{c_1, c_2 \dots c_n\}$, yorumları $W = \{w_1, w_2 \dots w_n\}$, birleştirerek giriş için $X = \{[CLS], W, [SEP], C\}$ vektörü oluşturulur.

CodeBERT hem doğal dil-kod çiftlerinin iki modlu verilerini hem de tek modlu verileri kullanmasına olanak tanıyarak, doğal dil ile kod arama ve kod

dokümantasyonu oluşturma gibi uygulamalarda performansı artırır. Bu sayede CodeBERT modeli çok çeşitli görevlerde iyi performans gösterebilmektedir.

Çalışmada CodeBERT için H^n çıktı vektörü, kod yorumları göz ardı edilerek üretilerek sınıflama aşamalarında kullanılmıştır.

2.2.3. GraphCodeBERT

GraphCodeBERT, kodun iç yapısı ve veri akış grafını dikkate alan bir ön eğitimli modeldir. Soyut sözdizimi ağacı gibi kodun sözdizimsel düzeydeki yapısını almak yerine ön eğitim için anlamsal düzeydeki veri akışı bilgilerinden yararlanır (Guo vd., 2020). Temelde BERT (Devlin vd., 2019) ve Çift Yönlü Dönüştürücü (Vaswani vd., 2017) yapılarını kullanır. GraphCodeBERT yorumları, kaynak kodunu ve değişkenler kümesini giriş vektörüne alır ve yerleştirmeye dönüştürür. Ek olarak, değişkenler arası veri akışını belirtmek özel bir konum belirleme matrisi kullanır.

Çalışmada öncelikle bu model için de H^n çıktı vektörü üretilerek kullanılmıştır. Ancak yapılan testlerde bu çıktıyı kullanmak kod kokusu tespiti için istenen sonuçları vermediğinden son gizli katman H^{n-1}'in durumu da ek olarak eğitimde kullanılmıştır.

2.2.4. Kod Yerleştirmelerinin Oluşturulması ve Ortak Ağa Verilmesi İçin Yapılan Ön İşlemler

Çalışma kapsamında tüm dil modelleri için gereken API'ler çalışır hale getirilmiştir. Önceden eğitilmiş modeller, Hugging Face servisi üzerinden temin edilmiştir. Her dil modelinin çalışması için gerekli ön işlemler gerçekleştirilmiştir. Kod örnekleri her zaman yorumlardan temizlenerek işleme alınmıştır.

Bütün dil modellerinde ortak olarak yapılan işlem, kod verilerini model için tanımlı tokenizer'dan geçirmektir. Her dil modeli için en iyi performansı veren tokenizer seçilmiştir. Dil modelleri için kullanılan önceden eğitilmiş modellerin Hugging Face kütüphanesindeki isimleri Çizelge 2.4'de verilmiştir.

Çizelge 2.4 Dil modelleri için HuggingFace kütüphanesindeki tanımlayıcı değerleri

Dil Modeli	Tokenizer	Model
BERT	sentence-transformers/bert-base-nli-mean-tokens	sentence-transformers/bert-base-nli-mean-tokens
CodeBERT	microsoft/codebert-base	microsoft/codebert-base

GraphCodeBERT	microsoft/graphcodebert-base	microsoft/graphcodebert-base

Çalışmada kullanılan modellerin hepsi 512 belirteç sabit olarak çalışırlar. Bu sebeple, bir kod örneği için tokenizer sonucunda elde edilen belirteç sayısı; bu miktardan fazla ise sondan kesilir (truncate), az ise boş belirteçler ile doldurulur (padding). Bu 512 belirteç, her dil modeli için temeli oluşturur.

GraphCodeBERT modelinde ek ön işlem olarak kod örneklerinin kontrol akış grafiği de üretilmiştir. Bunun için Tree-sitter isimli ayrıştırıcı (parser) oluşturma aracı kullanılmıştır. Bu kontrol akış grafiğini temsil eden veriler tokenizer'a verilmiştir.

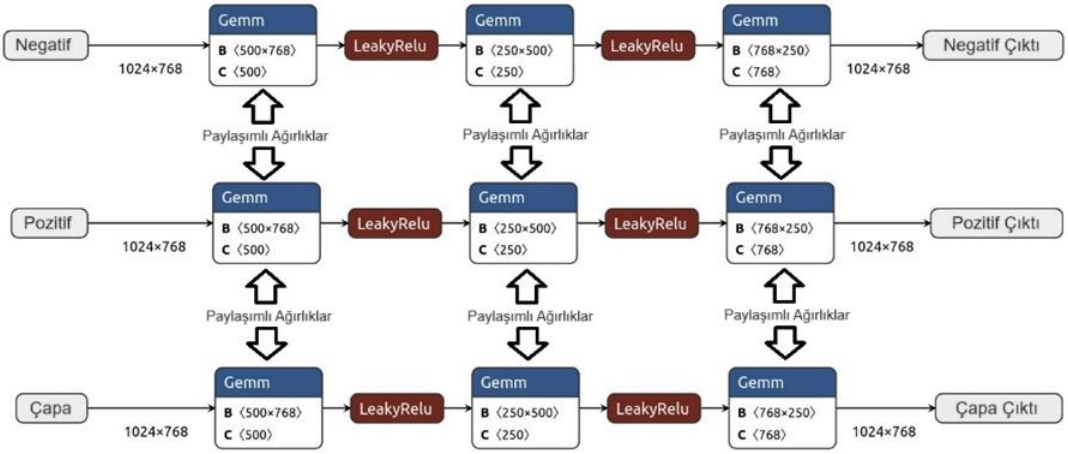
Bu modellerle üretilen kod yerleştirme vektörlerinin boyutları farklı olabilmektir. Her teknik için çıktı boyutu Çizelge 2.5'te verilmiştir. Ortak ağ yapısı faklı büyüklükteki ön eğitim sistemleri çıktılarıyla çalışacak şekilde ayarlanmıştır. Bu amaçla ortak ağ yapısına verilmeden önce kod yerleştirme sistemlerinin çıkışları bir düzleştirme (flatten) katmanından geçirilmektedir. Bu aşamada ilk katmanın vektör boyutu ilk katman giriş boyutu bir eşleştirilmektedir.

Çizelge 2.5 Kod yerleştirme modelleri için çıktı vektör boyutları

Ön eğitim modeli	Yerleştirme çıktı boyutu
BERT	768
CodeBERT	768
GraphCodeBERT	768
GraphCodeBERT (Son gizli katman)	320 x 768

2.3. ÜÇLÜ KAYIP İLE SINIFLAMA

Üçlü kayıp, kod yerleştirme vektörlerinin doğrudan nasıl optimize edeceğini öğrenmek için eğitilmiş bir DNN kullanır. Üçlü kayıp uygulamaları daha çok görüntü benzerliği için kullanıldığından evrişimli (convolutional) sinir ağları tercih edilmektedir (Schroff et al., 2015b). Ancak çalışmada kod benzerliği analiz edileceğinden PyTorch (Paszke vd., 2019) aracı ile çok katmanlı yoğun bir derin öğrenme ağı oluşturulmuştur; bkz. Şekil 2.2. Burada Gemm (General Matrix Multiplication), PyTorch'daki torch.Linear katmanını temsil eder. Yapılan deneyler sonucunda 3 katmanın performans açısından yeterli olduğu görülmüştür.



Şekil 2.2 Üçlü kayıp ağı yapısı.

Üçlü kaybın çalışma şekli şu şekildedir.

Girdi tensörleri x_1, x_2, x_3 ve bir marj değeri işleme alınır. Bu üç tensörün her birisi bir kod yerleştirmesi temsil eder. Bir üçlü a, p ve n (anchor, positive, negative; çapa, pozitif, negatif) değerlerinden oluşur. Bütün tensörlerin boyutları aynı olmalıdır. Bu tensörler ile aşağıdaki kayıp hesabı yapılır (PyTorch Contributors, 2024):

$$L(a,p,n) = \max\{d(a_i,p_i) - d(a_i,n_i) + \text{marj}, 0\} \quad (1)$$

Burada mesafe olan d fonksiyonu,

$$d(x_i,y_i) = \left|\left|x_i - y_i\right|\right|_\alpha \quad (2)$$

olarak verilir. Çalışmada α değeri 2, marj değeri ise 0.4 olarak kullanılmıştır.

Hızlı yakınsama, kalite ve çeşitlilik sağlamak için en ayırt edici üçlüleri seçmek çok önemlidir. Rastgele örnekleme, negatif kodları eşit şansla tekdüze bir dağılım olarak örneklemektedir. Basitliğine rağmen bu yöntemin, özellikle diğer negatif örnekleme stratejileriyle birleştirildiğinde, performansı artırmada etkili olduğu gösterilmiştir (Zhan vd., 2021). Bununla birlikte, negatif kodlar sorguya çok benzemiyorsa etkinliği sınırlı olabilir.

Çalışmada, bir koku türü içindeki kod yerleştirme vektörlerini eşleştirerek pozitif çiftler ve farklı koku türü örneklerindeki yerleştirmelerle eşleştirerek negatif çiftler oluşturmak için rastgele bir örnekleme stratejisi kullanıldı. Ancak, örnek sayısının düşük olduğu denemelerde, üçlü ağın kullanılmadığı durumlara kıyasla

performansta hafif bir düşüş olduğunu fark edildi. İstediğimiz sonucun düşük örnek sayılarından kaynaklanabileceğini düşünüldü. Bu zorluğun üstesinden gelmek için, aşağıdaki denklemi kullanarak tüm olası çaprazlamaları oluşturarak örnekleme boyutunu genişletildi.

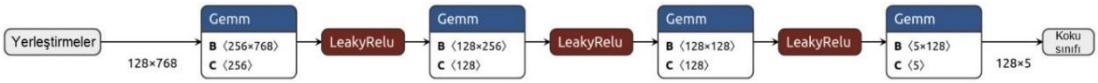
$$CE_1 x CE_2 \dots x CE_n = \{(ce_1 x ce_2 \dots ce_n) : ce_1 \in CE_1, ce_2 \in CE_2 \dots ce_n \in CE_n\}$$

Örnekler $[\![(CE]\!]_1 x CE_2 \dots x CE_n)_i , i \in random(1,2..m)$ indeks kümesi ile oluşturuldu. m kartezyen kümenin toplam örnek sayısını göstermektedir. Seçimin indeksler kullanılarak yapılması ve doğrudan yerleştirme çiftlerinin oluşturulmaması sayesinde kartezyen çarpımının oluşturduğu çok fazla sayıdaki örneğin performansı düşürmesi engellenmiştir.

2.4. SINIFLAMA AĞI

Çalışmanın başlangıcında kod yerleştirmeleri farklı derin öğrenme modelleri üzerinde denenmiştir. Daha sonra kullanılan model ve bu modele giriş verilerinin uyarlanma süreci standart hale getirilmiştir. Oluşturulan sistemde farklı hiper parametrelerin optimizasyonu denenmesi ve ölçüm değişkenlerinin konfüzyon matris ve ilgili grafiklerinin çıkartılması otomatik olarak gerçekleştirilmektedir. Bu sayede yeni kod temsillerinin denenmesinde hiper parametre belirleme işlemi ve sonuçları analiz etmek daha kolay bir şekilde gerçekleştirilebilmektedir.

CodeBERT için 128 örnek ve 768 özellik içeren örnek bir girdi Şekil 2.3'de gösterilmiştir. Gemm dana önce belirtildiği gibi PyTorch'ta doğrusal bir katman olan genel matris çarpımı anlamına gelir. Modelde sırasıyla 256-128-128-5 nöronlardan oluşan ve aralarında Leaky ReLU aktivasyonu bulunan 4 katman vardır. Katman ve nöron sayısı, tatmin edici bir performans elde edilene kadar denenerek seçilmiştir.



Şekil 2.3 Ortak ağ modeli

Model mimarisi belirleme, belirteç oluşturma ve eğitim süreçlerinde tahmin performansını en üst düzeye çıkarmak için ayarlanması gereken birçok hiper parametre bulunmaktadır (Svyatkovskiy vd., 2020). Bu hiper parametreler, öğrenme hızı, dönüştürücü katman sayısı ve gömme uzayının boyutu gibi sayısal veya

kullanılan optimizer gibi kategorik değerleri içerir. Dönüştürücü sinir ağlarının optimizasyonu ve iyi performans gösteren hiper parametrelerin seçimi karmaşık bir hesaplama problemidir ve yüksek boyutlu bir uzayda arama yapılmasını gerektirir (Swarna vd., 2023).

Geliştirilen sistemde tüm hiper parametreler dışarıdan JSON formatında bir dosya içerisinde sisteme verilmektedir. Çıktılar da aynı şekilde ortak dosyalara kaydedilmekte ve bu dosyaları görselleştirilmesi için de ortak bir uygulama kullanılmaktadır. Geliştirilen ortak sınıflama ağı üzerinde ızgara araması (grid search) işlemi gerçekleştirmiştir. Izgara aramasında aşağıdaki parametreler verilen sınırlar arasında adım adım arttırılmıştır:

Optimizer, ağırlıklar ve öğrenme oranları güncelleme kuralları, öğrenme oranları ve momentum stratejileri gibi sinir ağının niteliklerini ayarlayan bir fonksiyondur. Kayıp fonksiyonunun dik inişi yönünde ağırlıkları iteratif olarak günceller. Tez çalışmasında Stokastik Gradyan İnişi (SGD) ve Adam optimizerlarının performansları test edilmiştir.

Öğrenme oranı (*learning rate, LR*), gradyan optimizasyonu sırasında ağırlık güncellemelerinin boyutunu belirler. LR, $(10^{-4}, 10^{-5}, 10^{-6}, 10^{-7})$ değerleri içerisinden seçilmiştir.

Yığın boyutu (*batch size, BS*), modelin parametrelerini güncellemek için bir iterasyonda kullanılan eğitim örneklerinin sayısını ifade eder. Eğitimin hızını ve kararlılığını ve bellek kullanımını etkiler. Daha küçük yığın boyutlar daha sık güncellemeler sunar ancak gürültülü gradyanlara ve daha yavaş yakınsamaya neden olabilir. Daha büyük yığın boyutları daha yumuşak gradyanlar sağlar ancak daha fazla bellek ve daha yavaş hesaplama gerektirebilir. BS değerleri 16, 32, 48, 64 ve 80 arasından denenmiştir.

2.5. ÜÇLÜ KAYIP İLE KOD BENZERLİK ANALİZİ

Üçlü kaybın ikinci bir uygulaması olarak kod benzerlik analizi gerçekleştirilmiştir. Bu analiz içerisinde test kapsamında hazırlanan üçlü kayıp fonksiyonunun transfer öğrenme yöntemiyle kod benzerlik analizine uyarlanması yapılmıştır. Ayrıca tez kapsamında incelenen BERT tekniği de kod yerleştirmelerin üretilmesi aşamasında kullanılmıştır.

Kod benzerliği sistem içerisinde aynı işleve sahip kodların tekrar etmesiyle ortaya çıkan ve projemizde kullanılacak en önemli kokan kod türlerinden birisidir. Bu konuda erken dönemde yapılan çalışmalar, basit kod benzerlik analizi için etkili olan kodun sözdizimsel analizinde odaklanmıştır. Günümüzde, kod benzerliği tespiti için statik kod özelliklerine dayalı intihal tespit araçları yaygın olarak kullanılmaktadır. Ancak SonarQube dahil bu araçlar koddaki sıra değişimi ve değişken isim değişikliği gibi statik özellikle yoğunlaşmıştır. Farklı sözdizimlerinin aynı görevleri yerine getirebileceği durumlarda genellikle anlamsal benzerlikler tespit edilememektedir. Derin öğrenme alanındaki ilerlemelerle, AST ve kontrol akış çizelgesi gibi teknikler aracılığıyla daha kapsamlı ve anlamsal analiz mümkün hale gelmiştir. Bu da kod işlevselliğinin daha ayrıntılı bir şekilde anlaşılmasını sağlamıştır.

Kod benzerliğini tespiti için derin öğrenmenin kullanımı yenilikçi bir araştırma alanıdır. Çalışma kapsamında, kod benzerliğini tespit etmek için üçlü kayıp tabanlı yenilikçi bir derin öğrenme sistemi geliştirilmiştir.

Kod benzerlik analizinde Java kod parçacıklarını vektörel yerleştirme için ön eğitimli bir model olan CodeBERT tercih edilmiştir. Öğrenme aşamasından sonra, kod benzerliğini tespit etmek için eğitilmiş üçlü ağ ağırlıkları sınıflandırıcıya transfer edilmiştir. Önerilen sistemin etkinliği, üçlü kayıp entegrasyonu olan ve olmayan modellerin kayıp değerlerindeki azalma ve doğrulukta iyileşme açısından değerlendirilmiştir.

CodeBERT kullanılarak kod yerleştirme vektörlerinin çıkarılmasının ardından, modelin bu yerleştirmeleri kullanarak kod benzerliklerini tespit yeteneğini geliştirmek için yoğun bir katman mimarisi oluşturmuştur. Yerleştirme vektörleri ve etiketlerini benzer veya değil şeklinde sınıflamak için Gated Recurrent Unit tabanlı bir ağ modeli oluşturulmuştur.

ÜÇÜNCÜ BÖLÜM

3. BULGULAR

Modelleri eğitmek ve değerlendirmek için TÜBİTAK projesi kapsamında alınan Grafik İşlem Birimleri (GPU'lar) kullanıldı ve tüm deneyler 10752 CUDA çekirdeği, 336 Tensor çekirdeği, 84 RT Çekirdeği ve 48GB GDDR6 bellek ile donatılmış NVIDIA GPU RTX 6000 üzerinde gerçekleştirildi.

3.1. KOKAN KOD TESPİTİ İÇİN BULGULAR

BERT, CodeBERT ve GraphCODEBERT modellerinin doğruluk, kesinlik, duyarlılık ve F1-skor metriklerini kullanarak üçlü kayıp kullanılan ve kullanılmayan durumlarda performanslar incelendi. Ayrıca, performans artışının arkasındaki nedeni açıklamak için üçlü kayıp öncesi ve sonrası ağların sınıf dağılımlarını da araştırıldı. Sunulan sonuçların rasgele seçimlerle elde edilmesi için 5 kat (fold) doğrulama kullanılmıştır.

Bu çalışmada Çizelge 2.2'de verilen her koku için 500 adet metot kodu içeren bir veri seti kullanılmıştır. Veri kümesi oluşturulurken bazı kokan kod türlerinden 1000 adedin üzerinde bazılarından ise 500 civarında örnek toplanmıştır. Ancak dengesiz veri kümesi üzerinde yapılan analiz ve test işlemleri performans açısından oldukça düşük sonuçlar vermiştir. Bu yüzden örnek adedi en düşük olan sınıfa yakın, 500 sayısı esas alınarak her sınıftan rastgele 500 örnek seçilmiştir. Toplam olarak 5 farklı kokuya sahip 2500 örnek ile çok sınıflı sınıflandırma yapılmıştır.

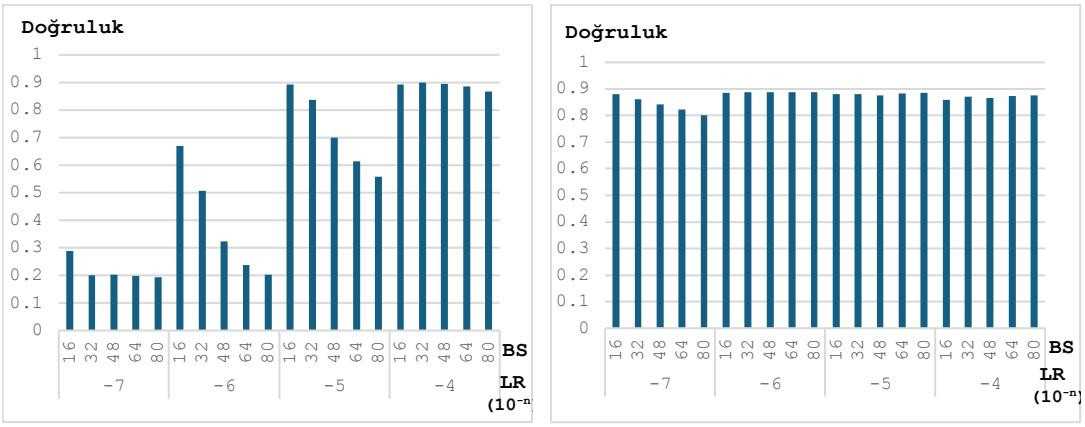
Modelin performans değerlendirmesi için doğruluk, kesinlik, duyarlılık, F1-Skor ve kayıp gibi değerlendirme ölçütleri kullanılmıştır. Sınıflandırma için en iyi kombinasyonu seçmek amacıyla her modelin doğruluğunu farklı parametre ayarlarıyla değerlendirerek çeşitli yerleştirme modelleri için en uygun hiper parametreler belirlenmiştir. Hiper parametre optimizasyonu için elde edilen sonuçlar

Şekil 3.1'de verilmiştir. Ayrıca, tutarlı ve uygun sayıda yinelemenin (epoch) seçilmesini sağlayarak, yakınsamaya ulaşılıncaya kadar yinelemeler boyunca her algoritmanın aktivasyon fonksiyonu çıktıları izlendi.

Çizelge 3.1 Her hiper parametre için denenecek olan değerlerin kümesi

Hiper Parametre	Denenme Kümesi
Optimizer	{ Adam, SGD}
LR	$\{10^k \mid k \in \{-7, -6, -5, -4\}\}$
BS	$\{ 16, 32, 48, 64, 80 \}$

Hiper parametrelerin deneme kümeleri Çizelge 3.1'de verilmiştir. Izgara araması ile her bir optimazyon çalıştırmasında bütün mümkün kombinasyonlar denenmiştir. Bu da her çalıştırmada modelin 40 kere çalışmasını gerektirmiştir.



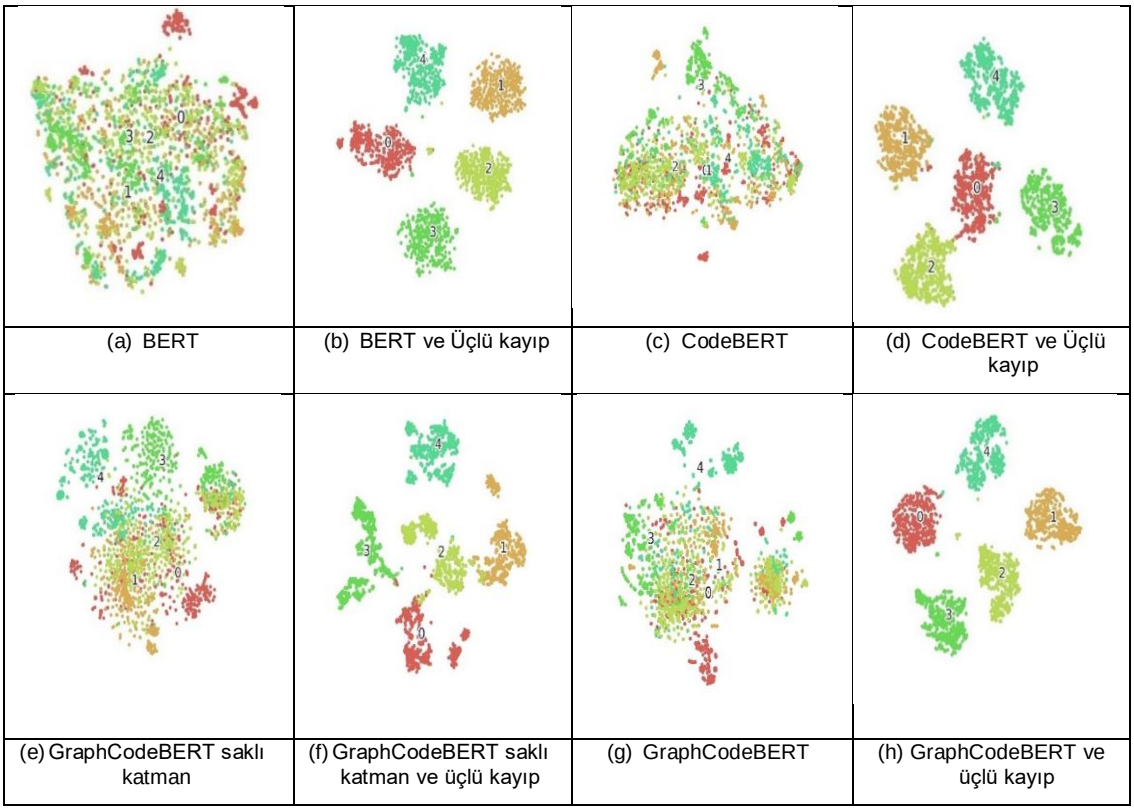
(a) SGD (b) Adam

Şekil 3.1 Farklı hiper parametreler için doğruluk hesapları

Şekil 3.1'de görüldüğü üzere Adam optimizer, SGD'den daha tutarlı sonuçlar vermiştir. Ancak en yüksek başarıyı %89,92 ile SGD optimizer sağlamıştır. Sonuç olarak modellerde kullanılan ortak parametreler; SGD optimizer, 10^{-4} LR ve 32 BS olarak seçilmiştir.

Sağlanan iyileşmenin sebeplerinin anlaşılması ve görselleştirilmesi için her veri noktasına iki boyutlu bir haritada bir konum vererek yüksek boyutlu verileri görselleştiren t-SNE (t-distributed stochastic neighbor embedding, t-dağıtılmış stokastik komşu yerleştirme) tekniğini kullanıldı (Van Der Maaten & Hinton, 2008). Şekil 3.2 a-h, sınıfa ve veri türüne göre farklı renklere sahip normal ve geliştirilmiş yerleştirmeler için t-SNE grafiklerini göstermektedir. Burada 0-4 aralığındaki sayılar

Çizelge 2.3'te belirtildiği üzere sırasıyla java:S100, java:S1161, java:S1452, java:S119, java:S1172 kokularını temsil etmektedir. Grafikler, verilerin her yönde karmaşık bir dağılıma sahip olduğunu ve ifadeler arası ilişkilerin daha düşük olduğunu göstermektedir. Üçlü kayıp, küme kalitesini artırarak her koku türü için yüksek küme uyumu sağlamıştır.

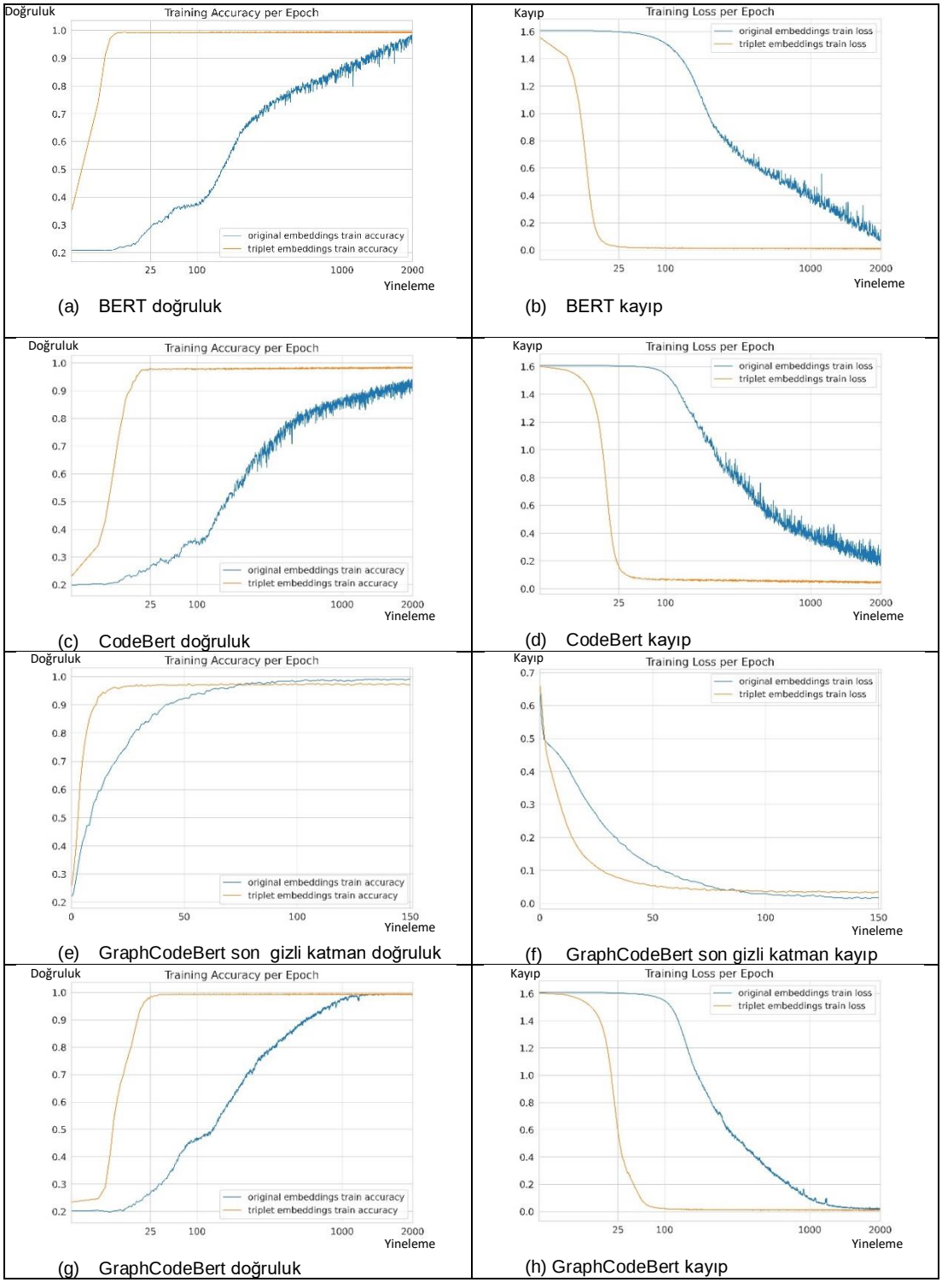


(a) BERT | (b) BERT ve Üçlü kayıp | (c) CodeBERT | (d) CodeBERT ve Üçlü kayıp

(e) GraphCodeBERT saklı katman | (f) GraphCodeBERT saklı katman ve üçlü kayıp | (g) GraphCodeBERT | (h) GraphCodeBERT ve üçlü kayıp

Şekil 3.2 Yerleştirmelerdeki iyileşme düzeyleri

Şekil 3.3 a-h, değerlendirilen her model için üçlü kayıp bazlı iyileştirmeler içeren ve içermeyen test doğruluğunu ve yineleme sayısını gösterir. Üçlü kayıp temelli modelin yüksek yakınsama oranına ve düşük kayıp değerlere sahip olduğu görülmektedir. Bu iyileştirmeler farklı yerleştirme modelleri için benzer özellikler sergilemiştir. Ön eğitim modellerinin çıktısı olan yerleştirmelerin doğrudan kullanımı, doğrulama sürecinin 2000'inci yinelemesinde en iyi sonuçlarını elde ederek uzun bir süreçte öğrenmeye devam etmiştir.



Şekil 3.3 Farklı algoritmalar için yinelemeye göre doğruluk ve kayıp grafikleri

Çizelge 3.2, tüm modellerin performans ölçümlerini özetlemektedir. Deneyler, önerilen yaklaşımın her model için doğruluk, kesinlik (precision), hatırlama (recall) ve F1 puanında önemli bir iyileşmeye yol açtığını ortaya

koymaktadır. Üçlü kayıp olmadan GraphCodeBERT'in son gizli durumu en iyi performansı verir. Beklenmeyen bir sonuç olarak, üçlü kaybı kullanırken, metin analizi için tasarlanmış bir model olan BERT, kod için tasarlanmış modeller ile boy ölçüşecek bir performans sağlamıştır. Bu da metin tabanlı geniş modellerin kod analizindeki gücünü göstermektedir.

Çizelge 3.2 Tüm modeller için test performans ölçümlerinin özeti

Model	Doğruluk(%)		Kesinlik(%)		Duyarlılık(%)		F1-Skor(%)	
	Temel model	Üçlü Kayıp	Temel model	Üçlü Kayıp	Temel model	Üçlü Kayıp	Temel model	Üçlü Kayıp
BERT	80,44	98,76	80,49	98,76	80,44	98,76	80,45	98,76
CodeBERT	86,00	97,76	85,99	97,77	86,00	97,76	85,99	97,76
GraphCodeBert	81,84	**98,80**	81,78	**98,80**	81,84	**98,80**	81,80	**98,80**
GraphCodeBert son saklı katman	**89,92**	97,36	**90,15**	97,36	**89,92**	97,36	**89,95**	97,36

Deneysel sonuçlar, önerilen üçlü kayıt tabanlı yaklaşımın çeşitli ön eğitimli kod yerleştirme vektörleri ile birlikte kullanıldığında kokan kod sınıflarının tespitinde daha iyi ve etkin sonuçlar sağladığını göstermiştir. Yapılan çalışma, yeni kod yerleştirme yöntemleri tasarlamak yerine bir ön işlem olarak iyi üçlü kaybın kullanımın model performansı üzerinde önemli bir etki oluşturabileceğini ve kod yerleştirmeye yönelik mimari yeniliklerle karşılaştırılabilir olduğu gösterilmiştir.

3.2. KOD BENZERLİĞİ İÇİN BULGULAR

Üçlü kayıp ile benzerlik analizinde kaynak kodu hırsızlığı tespiti için GitHub'dan elde edilmiştir (van Schwartzenberg, 2020). Veri seti, başlangıç düzeyindeki yedi programlama değerlendirme görevini kapsayan 467 Java kod dosyasından oluşur. Veri kümesi, eğitim-test bölme fonksiyonu kullanılarak sırasıyla %80 ve %20 dağılımlarla eğitim ve test kümelerine ayrılmıştır. Sonuçlar CodeBERT'in doğrudan kullanımı ve CodeBERT + Üçlü Kayıp sistemi için değerlendirilmiş ve karşılaştırılmıştır.

Bu çalışmanın sonuçları Çizelge 3.3'de verilmiştir. Görüldüğü üzere bütün ölçümlerde üçlü kayıp performansı iyileştirmiştir.

Çizelge 3.3 Kod benzerliği test sonuçları

Model	Test Kaybı	Test doğruluğu (%)	Kesinlik (%)	Duyarlılık (%)	F1-Skor (%)
CodeBERT	0,1040	95,59	96,35	98,50	97,42
CodeBERT + Üçlü kayıp	**0,0548**	**97,73**	**98,14**	**99,24**	**98,69**

Bu çalışma ICAETA-2024 konferansında “Analysis of Code Similarity with Triplet Loss-Based Deep Learning System” başlığıyla sunulmuştur.

SONUÇ

Kayıp cezası tabanlı yaklaşımla karşılaştırmalı analiz sonuçları, üçlü kayıp tabanlı iyileştirme kullanmanın farklı yerleştirme modelleri için %7-%19 daha iyi performans sağladığını göstermektedir. Şekil 3.2'deki t-SNE grafikleri incelendiğinde, performans artışının ana kaynağının üçlü tabanlı iyileştirmeler uygulandığında sınıflar arasındaki dağılımın daha düzgün hale gelmesi olduğu görülmektedir. Bu sonuçlardan yola çıkarak, önerilen sistemin kod sınıflandırma görevleri için bir ön adım olarak kullanılabileceği düşünülmektedir. Bu sistem derin öğrenme tekniklerinin statik kod analiz araçlarına entegre edilmesi için kullanılabilir.

Tüm modellerin artan doğruluğu, üçlü kayıp tabanlı karşılaştırmalı öğrenmenin, kod sınıflandırma görevlerinde kullanılmak üzere önceden eğitilmiş kod yerleştirmelerin geliştirilmesinde genelleştirilebileceğini kanıtlamıştır. Sonuçlar ayrıca önerilen modellerin diğer önceden eğitilmiş yerleştirme tekniklerine uyarlanabilirliğini de göstermektedir. Bu yaklaşım, kod ve metin analizi için daha basit ve daha hızlı tekniklerin kullanılmasına olanak sağlamaktadır.

Önerilen sistem doğrudan derin öğrenmeye dayalı kod kalite kontrol araçlarının geliştirilmesi için de bir potansiyel oluşturmaktadır. Derin öğrenmenin sadece kod metinlerinden ve bu metinlerden çıkartılan özniteliklerden (AST, kontrol akış şeması vs. gibi) yapılabilmesi; kodun derlenebilir durumda olmasını gerektiren, bu yüzden sadece tümüyle derlenen projeler çapında çalışabilen bazı statik kod analiz araçlarına göre önemli bir üstünlüktür. Yapılan çalışmada kullanılan derin öğrenme tekniğinin kod benzerliğinin belirleme alanındaki avantajı da semantik benzerliği yüksek başarı ile tespit edebilmesidir. Statik kod analizi araçları bu tür anlamsal benzerlikleri tespit etmede yetersiz kalmaktadır.

Yerleştirme modelinin karmaşıklığı arttıkça üçlü kaybın etkinliğinin azaldığı da gözlemlenmiştir. Üçlü kayıp kullanımı, metin analizi için tasarlanmış olan BERT modelinin performansını karmaşık kod analiz modelleri seviyesine çıkarmıştır.

GEÇERLİLİĞE YÖNELİK TEHDİTLER

Sadece kokan kod tespiti görevi için önceden eğitilmiş yerleştirme modelleri değerlendirilmiştir. Modellerin genellenebilirliğini artırmak için, karşılaştırmalı öğrenme ile geliştirilmiş yerleştirmeler, klon tespiti, kod üretimi, yazılım kalitesi

değerlendirmesi ve kod incelemesi gibi diğer kod analizi görevlerinde de değerlendirilmelidir.

Hiper parametreler önceden eğitilmiş modellerin performansı üzerinde önemli bir etkiye sahiptir ve iç tutarlılığı etkileyebilir. Farklı hiper parametreler farklı tekniklerde daha iyi sonuçlar verebilir. Sistemleri doğrudan karşılaştırmak için bu etki göz ardı edilmiş ve tüm teknikler için önceden belirlenmiş ve GraphCodeBERT için iyileştirilmiş değerler kullanılmıştır.

Model testleri, Java ile yazılmış GitHub depolarından yöntem düzeyinde toplanan veriler kullanılarak gerçekleştirildiği için veri kümesi boyutu nispeten küçük kalmıştır. Veri setini genişletmek ve farklı yazılım dilleriyle denemeler yapmak, dış geçerliliği artırmaya ve sistemi genelleştirmeye yardımcı olabilir.

GELECEK ÇALIŞMALAR

Bu çalışmada, kod sınıflandırma görevi için farklı yerleştirme türlerini optimize eden, karşılaştırmalı öğrenmeye dayalı bir yaklaşım önerilmiştir. Önerilen yöntemin etkinliğini değerlendirmek için iki aşamalı bir sistem geliştirilmiştir. Öncelikle, yerleştirmeler üçlü kayıp tabanlı bir ağ ile iyileştirilmiş ve daha sonra bu iyileştirmenin yerleştirmeler bir sınıflandırıcı kullanılarak kokan kodların sınıflandırılması işleminde test edilmiştir. Sonuçlar, önerilen yaklaşımın çeşitli ön eğitimli yerleştirmeler için doğrulukla ilgili metriklerde daha iyi sonuçlar sağladığını göstermektedir.

Bu çalışmanın önemli bir sonucu, kod temsillerinin performansını artırmak için ön işleme yöntemlerinin kullanılabileceğinin göstermesidir. Ön işleme yöntemlerinin nasıl kullanılabileceği de ele alınmıştır. Bu amaçla, kod yerleştirmeleri için artırma ve kayıp cezası gibi ön işleme tekniklerinin denenmesi de mümkündür.

Bu alanda daha iyi sonuçlar elde etmek için gelecekte araştırılabilecek konular mevcuttur. Öncelikli bir araştırma konusu, ön işlemleri kod analizine dayalı yerleştirmelere göre yeniden yapılandırmaktır. Bu amaçla farklı kayıp fonksiyonları ve üçlü ağ modelleri geliştirilebilir. Bir diğer araştırma alanı ise zıt öğrenmenin önceden eğitilmiş modelin iç yapısına entegre edilmesi konusudur. Böylece tek bir ağ modeliyle daha performanslı bir öğrenme süreci tasarlanabilir.

KAYNAKÇA

Alazba, A., & Aljamaan, H. (2021). Code smell detection using feature selection and stacking ensemble: An empirical investigation. Information and Software Technology, 138. https://doi.org/10.1016/j.infsof.2021.106648

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2017). A Survey of Machine Learning for Big Code and Naturalness. http://arxiv.org/abs/1709.06182

Alon, U., Brody, S., Levy, O., & Yahav, E. (2018). code2seq: Generating Sequences from Structured Representations of Code. http://arxiv.org/abs/1808.01400

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2018). code2vec: Learning Distributed Representations of Code. http://arxiv.org/abs/1803.09473

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). Code2Vec: Learning Distributed Representations of Code. Proceedings of the ACM on Programming Languages, 3(POPL), 1-29. https://doi.org/10.1145/3290353

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data, 8(1). https://doi.org/10.1186/s40537-021-00444-8

Ben-Nun, T., Jakobovits, A. S., & Hoefler, T. (2018). Neural code comprehension: A learnable representation of code semantics. Advances in Neural Information Processing Systems, 2018-Decem(NeurIPS), 3585-3597.

Brown, R., & Greer, D. (2023). Software Code Smells and Defects: An Empirical Investigation. Içinde H. Kaindl, M. Mannion, & L. Maciaszek (Ed.), Proceedıngs Of The 18th Internatıonal Conference On Evaluatıon Of Novel Approaches To Software Engıneerıng, Enase 2023 (ss. 570-580). https://doi.org/10.5220/0011974500003464

Bui, N. D. Q., Yu, Y., & Jiang, L. (2021). Self-Supervised Contrastive Learning for Code Retrieval and Summarization via Semantic-Preserving Transformations. SIGIR 2021 - Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, 511-521. https://doi.org/10.1145/3404835.3462840

Büyük, O. O., & Nizam, A. (2023). Deep learning with class-level abstract syntax tree and code histories for detecting code modification requirements. Journal of Systems and Software, 206. https://doi.org/10.1016/j.jss.2023.111851

Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). A Simple Framework for Contrastive Learning of Visual Representations. https://github.com/google-research/simclr.

Cunningham, W. (1992). The WyCash portfolio management system. SIGPLAN OOPS Mess., 4(2), 29-30. https://doi.org/10.1145/157710.157715

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference, 1(Mlm), 4171-4186.

Dong, X., & Shen, J. (2018). Triplet Loss in Siamese Network for Object Tracking. European Conference on Computer Vision. https://api.semanticscholar.org/CorpusID:52959623

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. Findings of the Association for Computational Linguistics Findings of ACL: EMNLP 2020, 1536-1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

Gao, T., Yao, X., & Chen, D. (2021). SimCSE: Simple Contrastive Learning of Sentence Embeddings. http://arxiv.org/abs/2104.08821

Geisler, S., Li, Y., Mankowitz, D., Cemgil, A. T., Günnemann, S., & Paduraru, C. (2023). Transformers Meet Directed Graphs. Proceedings of Machine Learning Research, 202(c), 11144-11172.

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C., Drain, D., Sundaresan, N., Yin, J., Jiang, D., & Zhou, M. (2020). GraphCodeBERT: Pre-training Code Representations with Data Flow. http://arxiv.org/abs/2009.08366

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. Neural computation, 9, 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2023). Large Language Models for Software Engineering: A Systematic Literature Review. 1(1), 1-62.

Huang, J., Zhao, J., Rong, Y., Guo, Y., He, Y., & Chen, H. (2023). Code Representation Pre-training with Complements from Program Executions. 1, 1-14.

Li, H., Miao, C., Leung, C., Huang, Y., Huang, Y., Zhang, H., & Wang, Y. (2022). Exploring Representation-Level Augmentation for Code Search. Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, 4924-4936. https://doi.org/10.18653/v1/2022.emnlp-main.327

Lin, J., Nogueira, R., & Yates, A. (2020). Pretrained Transformers for Text Ranking: BERT and Beyond. http://arxiv.org/abs/2010.06467

Liu, L., Nguyen, H., Karypis, G., & Sengamedu, S. (2021). Universal Representation for Code. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12714 LNAI, 16-28. https://doi.org/10.1007/978-3-030-75768-7_2

Mäntylä, M., Vanhanen, J., & Lassenius, C. (2003). A Taxonomy and an Initial Empirical Study of Bad Smells in Code.

Martin Fowler, Kent Beck, & John Brant. (1999). Refactoring: Improving the Design of Existing Code. Addison-Wesley.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. http://arxiv.org/abs/1301.3781

Niu, C., Li, C., Ng, V., Chen, D., Ge, J., & Luo, B. (2023). An Empirical Comparison of Pre-Trained Models of Source Code. http://arxiv.org/abs/2302.04026

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Içinde Advances in Neural Information Processing Systems 32 (ss. 8024-8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

PyTorch Contributors. (2024, Haziran 12). TripletMarginLoss. https://pytorch.org/docs/stable/generated/torch.nn.TripletMarginLoss.html

Rasool, G., & Arshad, Z. (2015). A review of code smell mining techniques. Journal of Software: Evolution and Process, 27, 867-895. https://api.semanticscholar.org/CorpusID:38260067

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 07-12-June, 815-823. https://doi.org/10.1109/CVPR.2015.7298682

Shi, K., Lu, Y., Chang, J., & Wei, Z. (2020). PathPair2Vec: An AST path pair-based code representation method for defect prediction. Journal of Computer Languages, 59. https://doi.org/10.1016/j.cola.2020.100979

Sui, Y., Cheng, X., Zhang, G., & Wang, H. (2020). Flow2Vec: Value-flow-based precise code embedding. Proceedings of the ACM on Programming Languages, 4(OOPSLA). https://doi.org/10.1145/3428301

Tufano, M., Palomba, F., Bavota, G., Olivetox, R., Di Penta, M., De Lucia, A., & Poshyvanyk, D. (2015). When and why your code starts to smell bad. Proceedings - International Conference on Software Engineering, 1, 403-414. https://doi.org/10.1109/ICSE.2015.59

Van Der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. Içinde Journal of Machine Learning Research (C. 9).

van Schwartzenberg, S. (2020). Security Study with the Use of Known Vulnerabilities in Github.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. http://arxiv.org/abs/1706.03762

Yahav, E., & Levy, O. (2019). Code2Seq:Generatıng Sequences From Structured Representatıons Of Code. ICLR 2019 CODE2SEQ:, 1, 1-19.

Zhan, J., Mao, J., Liu, Y., Guo, J., Zhang, M., & Ma, S. (2021). Optimizing Dense Retrieval Model Training with Hard Negatives. http://arxiv.org/abs/2104.08051

Zhang, F., Peng, M., Shen, Y., & Wu, Q. (2024). Hierarchical features extraction and data reorganization for code search. Journal of Systems and Software, 208(September 2023), 111896. https://doi.org/10.1016/j.jss.2023.111896

Zhuo, T. Y., Yang, Z., Sun, Z., Wang, Y., Li, L., Du, X., Xing, Z., & Lo, D. (2023). Data Augmentation Approaches for Source Code Models: A Survey. Ml. http://arxiv.org/abs/2305.19915