İZMİR BAKIRÇAY ÜNİVERSİTESİ
MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

DOĞAL DİL İŞLEME
DÖNEM PROJESİ RAPORU

DR. ÖĞR. ZEKERİYA ANIL GÜVEN

Sultan Sarızeybek
Gamze Nurlu
İrem Kumlu

İÇİNDEKİLER

ŞEKİLLER TABLOSU

1. GİRİŞ

Metin üretme, bir yapay zekâ sisteminin insan dili kalıplarını ve stillerini taklit ederek yazılı içerik ürettiği bir süreçtir. Süreç, doğal insan iletişimine benzeyen tutarlı ve anlamlı metinler üretmeyi içerir. Metin oluşturma, doğal dil işleme, içerik oluşturma, müşteri hizmetleri ve kodlama yardımı dahil olmak üzere çeşitli alanlarda büyük önem kazanmıştır.

Metin oluşturma, giriş verilerini işlemek ve çıktı metni oluşturmak için algoritmalar ve dil modelleri kullanarak çalışır. Kalıpları, dilbilgisini ve bağlamsal bilgileri öğrenmek için yapay zekâ modellerinin büyük metin veri kümeleri üzerinde eğitilmesini içerir. Bu modeller daha sonra öğrenilen bu bilgiyi, verilen istemlere veya koşullara dayalı olarak yeni metin oluşturmak için kullanır.

Metin oluşturmanın merkezinde, GPT (Generative Pre-trained Transformers) ve Google'ın PaLM'si gibi internetten gelen büyük miktarda metin verisi üzerinde eğitilmiş dil modelleri bulunur. Bu modeller, cümlelerin yapısını anlamak ve tutarlı ve bağlamsal olarak alakalı metinler oluşturmak için derin öğrenme tekniklerini, özellikle de sinir ağlarını kullanır.

Metin oluşturma süreci sırasında yapay zekâ modeli, bir cümle veya anahtar kelime gibi bir girdi alır ve öğrenilen bilgiyi, sonraki en muhtemel kelimeleri veya kelime öbeklerini tahmin etmek için kullanır. Model, istenen uzunluk veya koşul karşılanana kadar bağlam ve tutarlılığı birleştiren metin üretmeye devam eder.
Bu projenin temel amacı, en son teknolojiye sahip dil modellerini kullanarak metin oluşturma yeteneklerini keşfetmektir. Bunu başarmak için, metin oluşturmaya yönelik güçlü bir model olan PeftModel'i kullandık ve gamzeyy/mistral-7b-dolly modelini kullanarak fine tune (ince ayar) yaptık. Amacımız tutarlı ve kullanıcıdan alınan girdilerle alakalı, kaliteli ve bağlama duyarlı yeni metinler oluşturmaktır/üretmektir.

1.1 Metin üretme terminolojisi

Metin Üretme: Yapay zekâ modellerini kullanarak tutarlı ve bağlamsal olarak uygun metin içeriği oluşturma süreci olarak tanımlanır.

Doğal Dil İşleme (NLP): Doğal dil işleme (NLP), bilgisayarlara insan dilini yorumlama, işleme ve anlama yeteneği veren bir makine öğrenimi teknolojisidir.

Hugging Face: NLP araştırmaları ve uygulamaları için ortak bir alan sağlayan, en son teknolojiye sahip doğal dil işleme modellerini geliştirmesi ve paylaşmasıyla tanınan bir platformdur.

Transformer Mimarisi: Sıralı verileri işlemek için tasarlanmış, özellikle doğal dil işleme görevlerinde etkili olan bir sinir ağı mimarisi.

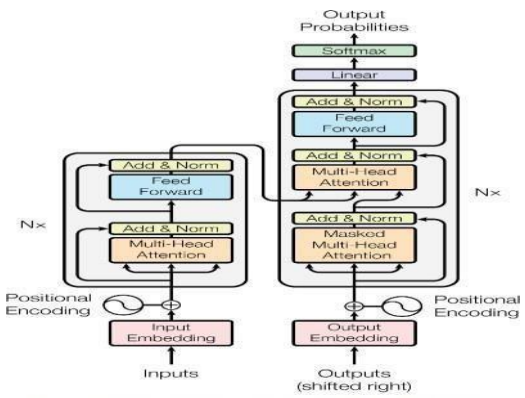


Figure 1: The Transformer - model architecture.

Tokenization: Metni belirteç adı verilen daha küçük birimlere ayırma işlemi. NLP' de metinverilerinin modellere beslenmesi için tokenizasyon esastır ve dilsel yapının daha iyi anlaşılmasına olanak tanır.

Aşırı Uyum ve Yetersiz Uyum: Makine öğreniminde sık karşılaşılan zorluklar. Aşırı uyum, bir model eğitim verilerinde iyi performans gösterdiğinde ancak yeni verilerde zayıf performans gösterdiğinde ortaya çıkarken, yetersiz uyum, modelin eğitim verilerindeki temel kalıpları yakalayamadığını gösterir.

GPT-2 Large: GPT-2 Large, özellikle doğal dil işleme görevleri için etkili bir sinir ağı tasarımı olan transformatör mimarisini temel alan bir dil modelidir. GPT-2 serisinin bir parçasıdır. Model, çok çeşitli metin verilerinden oluşan geniş bir külliyat üzerinde önceden eğitilmiştir ve 1,5 milyar parametreye sahiptir; bu, bir istem veya giriş verildiğinde tutarlı ve bağlamsal olarak alakalı metinler oluşturmasına olanak tanır. GPT-2 Large, dildeki uzun vadeli bağımlılıkları yakalama yeteneğiyle bilinir ve çeşitli doğal dil işleme uygulamaları ve yaratıcı metin oluşturma görevlerinde yaygın olarak kullanılır.

Fine Tune: Metin oluşturmaya yönelik, modelin tutarlı ve sağlanan girdiyle bağlamsal olarak alakalı metin oluşturacak şekilde eğitilmesini içerir. Model, eğitim verilerinde mevcut olan kalıplar ve stillerle uyumlu metinler oluşturmayı öğrenir.

Peft Tekniği: PEFT, kapsamlı kaynak ihtiyacını ve maliyeti en aza indirirken modellere fine tuning yapmak için tasarlanmış bir tekniktir. PEFT, model uyarlaması gerektiren alana özgü görevlerle uğraşırken en doğru seçimdir. Parametreli verimli ince ayar gerçekleştirmenin; Düşük Dereceli Parametre veya LoRA ve QLoRA en yaygın kullanılan yollardır.

Mistral Model: Tutarlı metin oluşturmak ve çeşitli doğal dil işleme görevlerini üstlenmek için geniş veri kümeleri üzerinde eğitilmiş güçlü bir yapay zekâ modelidir.

Quanzination: Büyük dil modelinde modelin boyutunu azaltarak performansı hızlandırmaktır. Örneğin; 32 bitlik model ağının 4 bite çevrilmesidir.

2. LİTERATÜR BİLGİSİ

Doğal Dil Üretimi (DDÜ), metin planlama ve metin üretme olmak üzere iki bölüme ayrılmaktadır. Metin planlama kısmında, kavramsal girdilerden metnin anlamsal tanımı üretilir. Daha sonra metin üretme sistemi, bu anlamsal tanımları gerçek bir metne dönüştürür [1]. Bir metinden farklı yeni metinler üretme, farklı bir konuda benzer metinler yazma veya aynı başlığı içeren yeni bir metin oluşturma gibi amaçlar için kullanılmaktadır [2]. Bu sayede, yapılandırılmış veri üzerinden yorum çıkarma gerçekleştirilebilmektedir. Bu uygulama alanında farklı dillerde, istatistiksel ve kural tabanlı birçok yöntem kullanılmıştır.

Mocan [3] ilköğretim düzeyindeki öğrencilerin okuduklarını anlama düzeylerinin geliştirilmesi amacıyla, metin işleme ve anlamaya dayalı soru soran bir sistem tasarımı üzerinde çalışmıştır.

Brown [4] istatistiksel teknikler ile sınıf bazlı n-gram modeli kullanarak İngilizce kelimeleri farklı sınıflara sözdizimsel veya semantik olarak gruplama çalışması gerçekleştirmiştir. Bu çalışma, tam ve anlamlı bir cümle oluşturmaktan ziyade, sözdizimsel veya anlamsal olarak kelime gruplarını bir küme altında toplama ile sınırlandırılmıştır.

Mairesse [5] sözdizimsel açıklama içermeyen, semantik kavramlardan bir biçim oluşturmak için, alana özgü veriler kullanarak dinamik Bayes ağını kullanan istatistiksel bir İngilizce dil üretici oluşturmuştur. Anlamsal olarak ilişkilendirilen yığınlardan yararlanarak oluşturulan cümleler her ne kadar farklı olsa da cümle sıralaması dikkate alınmadığından önemli ölçüde benzerlikler içermektedir.

Uchimoto [6] anahtar kelime veya başlıklardan cümleler üretmek için bir yöntem sunmuşlardır. İki aşamadan oluşan bu yöntemde geliştirdikleri model ile bilgi boşluğu veya kayıp kelimeleri bağımlılık ağaçları biçiminde ele alarak aday metin yapısı oluşturmuştur. Sözcükler arasındaki bağımlılık bilgilerinin yanı sıra n-gram bilgisini de dikkate alarak verilen anahtar kelimeler ile uygun cümleler elde etmiştir. Çalışmada daha fazla bilgiyi göz önüne alarak kurallar oluşturmak yerine, kısıtlı üretim kuralları ile çalışıldığı için hem doğal olmayan hem de gramer olarak uygun olmayan cümleler oluşmaktadır.

Tan Jiwei [7], bir belgenin önemli cümlelerini tanımlayan ve daha sonra benzer cümlelerden başlık üretmek için hiyerarşik cümle özetleme modelini önermiştir. Ancak bu çalışma ile kabaca oluşturulan özet cümlelerden yararlanarak başlık bilgisi elde edilmeye çalışılmıştır. Çalışmada en fazla 50 kelimeden ve bir veya iki cümleden oluşan özet cümleler oluşturulmuştur. Ancak bu özet cümleler tüm belgenin anlaşılmasına dayanan tamamen uçtan uca bir yöntem sunmamaktadır.

[8] Bauer ve arkadaşları kural tabanlı farklı türlerdeki doğal metinleri üreten bir sitem geliştirmişlerdir. Çalışmada doğal dil için kural temelli yaklaşımın uygun olduğunu ve problemin hızlı bir şekilde çözüleceği ileri sürmüşlerdir. Ancak, farklı dillerin sözdizimini açıklayan çeşitli kural setleri aracılığıyla ifade edebilmek, esnek bir makine oluşturmanın temelde karşılaşılan teknik zorluklardan biridir. Bu zorluk, çeşitli dillerde kullanılabilen ve cümleleri çok dilli bir şekilde ifade edebilen bir sistem geliştirmeyi engelleyen temel bir teknik meydan okumayı içermektedir.

3. UYGULAMA ALANLARI

Metin oluşturma, bir yapay zekâ sisteminin insan dili kalıplarını ve stillerini taklit ederek yazılı içerik ürettiği bir süreçtir. Süreç, doğal insan iletişimine benzeyen tutarlı ve anlamlı metinler üretmeyi içerir. Metin oluşturma, doğal dil işleme, içerik oluşturma, müşteri hizmetleri ve kodlama yardımı dahil olmak üzere çeşitli alanlarda büyük önem kazanmıştır.

Metin oluşturma, geniş bir uygulama yelpazesine sahip olan ve farklı sektörlerde çeşitli amaçlar için kullanılabilen bir teknolojidir. İşte metin oluşturmanın bazı detaylı uygulama alanları:

1. **Yaratıcı İçerik Üretimi**: Şairler, yazarlar, içerik oluşturucular veya sanatçılar için metin oluşturma modelleri, hikayeler, şiirler veya kreatif içerikler oluşturmak için kullanılabilir. Örneğin, yeni bir şarkı sözü veya kısa öykü yazmak için kullanılabilir.
2. **Yardım ve Destek Sistemleri**: Çeşitli platformlarda müşteri hizmetleri veya teknik destek sistemleri, kullanıcı sorularına otomatik cevaplar üretmek veya yardım dokümanları oluşturmak için metin oluşturma tekniklerini kullanabilir.
3. **Dil Çevirisi ve Uygulamaları**: Metin oluşturma modelleri, dil çevirisi ve çok dilli iletişimde kullanılabilir. Belirli bir dildeki içeriği anlık olarak başka bir dile çevirebilir.
4. **Hukuk ve Belgeleme**: Hukuk firmaları veya büyük kuruluşlar, hukuki belgeler, sözleşmeler veya raporlar oluşturmak için metin oluşturma tekniklerinden yararlanabilir.
5. **Eğitim ve Öğretim**: Eğitimciler, sınav soruları, öğrenci geribildirimi veya eğitim materyalleri oluşturmak için metin oluşturma modellerini kullanabilirler.
6. **Haber ve İçerik Üretimi**: Gazetecilik veya medya şirketleri, haber makaleleri, raporlar veya editoryal içerikler üretmek için metin oluşturma tekniklerinden yararlanabilir.
7. **Sağlık Sektörü**: Tıp alanında, hasta raporları, tıbbi belgeler ve tedavi planları gibi metinlerin oluşturulmasında kullanılabilir.
8. **Pazarlama ve Reklamcılık**: Pazarlama kampanyaları için içerik oluşturma, sosyal medya paylaşımları veya reklam metinleri oluşturmak için metin oluşturma teknikleri kullanılabilir.

Bu alanlar, metin oluşturmanın sadece birkaç örneğidir. Her bir alan, metin oluşturma tekniklerini farklı şekillerde kullanabilir ve bu teknoloji genellikle belirli ihtiyaçlara veya sektörlere uyacak şekilde özelleştirilebilir. Yaratıcılığı teşvik eden, iş akışlarını iyileştiren ve belirli görevleri otomatikleştiren birçok potansiyel uygulama bulunmaktadır.

4. ÖRNEK PROJELER

Muhtemelen günlük yaşamınızda metin oluşturma teknolojisiyle karşılaşmışsınızdır. İphone/Android'nizdeki otomatik doldurma özelliği, metin tamamlama, Google arama ve Google'ın Gmail'deki Akıllı Yazma özelliği yalnızca birkaç örnektir.

1- Sağlıkta Sentetik Metin Üretimi

İhtiyaç: Ulusal Sağlık Servisi'ndeki (National Health Service) metin verileri, uygun erişimle ilgili sorunlar ve serbest metin analizindeki incelikler nedeniyle oldukça az kullanılıyor. Üretilen sentetik tıbbi metnin çeşitli gerçekçi formatlarda sağlanması, potansiyeli belirleme ve yeniliği geliştirme konusunda daha fazla fırsat sağlayacaktır. Bu projede, sağlık yazılımı ve uygulamaları geliştiricileri için notlar ve hasta mektupları gibi kamuya açık, sentetik olarak oluşturulmuş tıbbi ücretsiz metinler oluşturmaya yönelik bir metodoloji oluşturmak amaçlanmış.

Proje aynı zamanda bu tür kaynakların uygun mahremiyeti koruma (ki bu önemli bir husustur) yeteneğini de değerlendirmeye alacak ve bunun, elde edilen metinlerin faydası ve kalitesi ile arasındaki dengeyi anlamaya çalışacaktır. Metinle çalışırken ve metin oluştururken modern büyük ölçekli dil modellerinin yaratıcılığını kullanma becerisi sağlamlık için özel olarak hazırlanmış bilgi sağlama ihtiyacıyla dengelenerek sağlık hizmetleri ortamlarındaki çözümlerde çok önemlidir.

Bu projede, metin oluşturmayı yönlendirmede, ipucu verme, gerçek bilgi kaynaklarını bütünleştiren teknikler veya ontolojiler gibi yapılandırılmış bilgi biçimlerinden modern üretken dil modelleri ile öğrenme gibi yeni yaklaşımları keşfetmeyi amaçlanmıştır.

2- ChatGPT

OpenAI'nin GPT-3.5 mimarisini temel alarak geliştirdiği bir dil modeliDİR. Doğal dil anlama ve oluşturma görevleri için tasarlanmış Üretken Ön Eğitimli Transformatör (GPT) serisinin bir parçasıdır. Yetenekleri arasında soruları yanıtlamak, sohbetlere katılmak, bilgi sağlamak ve çeşitli metin tabanlı görevlere yardımcı olmak yer alıyor.

3- GPT2-Şair

Kendi kendini denetleyen bir şekilde geniş bir İngilizce Şiir veri seti külliyatı üzerinde hassas şekilde ayarlanmış bir GPT-2 transformatör modelidir.

Model özellikle cümlelerdeki bir sonraki kelimeyi tahmin edecek şekilde eğitilmiş. Giriş dizileri, tanımlanmış uzunlukta sürekli metindir ve hedefler, bir jeton sağa kaydırılmış olarak aynı dizidir. Modelde, belirli bir token için tahminlerin gelecekteki tokenlar hariç olmak üzere yalnızca önceki tokenlardan gelen girdileri kullandığından emin olmak için dahili olarak bir maske mekanizması kullanılmıştır.

Bu kendi kendini denetleyen eğitim yaklaşımı, GPT2-Poet'in İngilizce dilinin nüanslarına dair içsel bir anlayış geliştirmesine olanak tanıyarak şiir oluşturma ve potansiyel olarak sonraki görevler için faydalı özellikleri çıkarma konusunda yetkin olmasını sağlamıştır.

4- Otomatik Hikâye Oluşturma: GPT-2 Kullanarak İngilizce Çocuk Hikayesi Oluşturma

[9] Bu çalışma, Üretken Önceden Eğitimli Transformer 2 (GPT-2), önceden eğitilmiş dil modellerinin, özellikle tutarlı metin oluşturma başta olmak üzere çeşitli görevlerdeki büyük ölçüdeki etkinliğini göstermiştir. Otomatik öykü oluşturma, geçmişte nadiren çalışılan büyük bir araştırma alanını temsil etmektedir. Ayrıca yapay zekâ alanında da önemli bir konu olması sebebiyle bu çalışmada, okuma becerilerini artırmak için Üretken Önceden Eğitimli Transformer 2'yi temel alan otomatik çocuk öyküleri oluşturmaya yönelik yeni bir yaklaşım sunulmuştur. Uygulamada, Hugging Face'in ünlü Transformers kütüphanesinin etrafına sarmalayıcı gibi inşa edilen Simple Transformers kütüphanesini kullanılmıştır. Bu, Transformer modellerinin çok hızlı bir şekilde eğitilmesine ve değerlendirilmesine olanak tanımıştır.

5. TÜRKÇE İÇİN YAPILAN ÇALIŞMALAR

[10] Kutlugün ve şirin çalışmalarında yıldız teknik üniversitesinin kemik doğal dil işleme grubunun sağladığı 42 bin haber isimli veri kümesin genel kategorisinde 6673 adetten oluşan haber içerikli metinlerden oluşan veri seti üzerinde çalışmışlardır. Çalışmanın temelinde istatiksel yaklaşımı uygulamış olup kısmi ve basit kural tabanlı yaklaşımdan da faydalanarak Türkçe metin üretme işlemi gerçekleştirmiş ve bu sayede degişken uzunlukta cümleler elde edilmiştir. Yaptığı analizde bigram ifadelerle yapılan metin üretme işleminde sayı olarak daha fazla yeni cümleler üretilmesine rağmen, anlam açısından daha düşük bağlamda üretilmiştir. Sondan iki bağlı trigram ifadelerde daha az cümle üretilmesine rağmen daha anlamsal bağı kuvvetli yeni cümleler oluşturulmuştur. Bu yöntem sistem başarımının daha yüksek seviyede olduğu tespit edilmiştir.

[11] Özdemir ve Amasyalı çalışmalarında ilk Türkçe hayat bilgisi veri tabanı(CSdb) kullanarak yeni metinler ürütme alanında çalışmalar yapmışlardır. Veri tabanında birbiriylekelime bazında benzerlik olmayan ama anlamca yakın olan metinlerin bulunmasında kullanılabilirliği gösterilmiştir. Gramer yapısı ve eşanlamlı kelimelerin yer değiştirmesi ile çalışma sonuçlanmakta olup performans sistemin veri tabanına bağımlı olduğu için veri tabanında iyileştirmeler yapılması içeriğin zenginleştirilmesi gerekmektedir.

[12] Adalı ve Erenler çalışmalarında bir hasta veri tabanından alınan verileri Türkçe dilininkurallarına uygun şekilde dizerek anlamlı ve dilbilgisi kurallarına uygun raporlar oluşturmaktadır. Çalışmada gramer oluşturmada ilişkisel veri tabanından faydalanmış olup, Türkçe gramer kuralları üzerinde çalışılıp Türkçe heceleme kuralları için bir sonlu durum makinesi oluşturulmuş ve fonemleri doğru olacak şekilde heceleme yapacak fonksiyon geliştirilmiştir. Ancak çalışmada kullanılan yöntem için oluşturulan gramer yapısı kullanılan veri tabanıyla sınırlı kalmaktadır.

6. PROJE FİKRİ

Blog Nedir?

Blog kelimesinin Türkçede karşılığı olan “ağ günlüğü” ifadesi, pek yaygın kullanılmıyor olsa da bu tip internet sitelerinin işlevini net şekilde belirtir. Zira genel itibariyle bloglar, klasik günlüklerin dijital ortama taşınmış, teknolojiye ayak uydurmuş hâli olarak tanımlanabilir.

Blog siteleri; metin, görsel, video ya da fotoğraf gibi içeriklerin paylaşılabildiği web tabanlı ağlardır. Bu tip sitelerde gönderiler genellikle, günlük formatını andırır şekilde güncelden daha eski içeriklere doğru sıralanır. **WordPress, Blogger, Tumblr ve Medium** gibi servisler üzerinden blog oluşturmak ücretsiz ve kolaydır.

Blog post, çevrimiçi bir platformda yazılan ve okuyuculara bilgi, deneyim, düşünceler veya eğlence sağlamak amacıyla yayınlanan bir yazıdır. Bir blog postunda bir konu

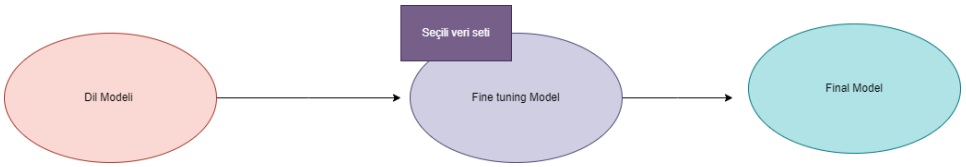
hakkında derinlemesine bilgi sunulabilir, bir deneyim paylaşılabilir veya bir tartışma başlatılabilir. Blog postları genellikle güncel ve ilgi çekici bir şekilde yazılır ve belirli bir hedef kitleye yöneliktir.

Blog postları, birçok farklı alanda faydalar sağlar. İlk olarak, blog postları insanların bilgi ve deneyimlerini paylaşmalarına olanak tanır. Ayrıca, blog postları, markalar, şirketler ve genel kuruluşlar için etkili araçlardır. Kuruluşun web sitesinin organik trafiğini artırabilir ve markayı arama motorlarında daha görünür kılarak SEO (Arama Motoru Optimizasyonu) stratejilerine katkıda bulunabilir.

PROJE GERÇEKLEŞTİRİMİ:

Fine Tuning:

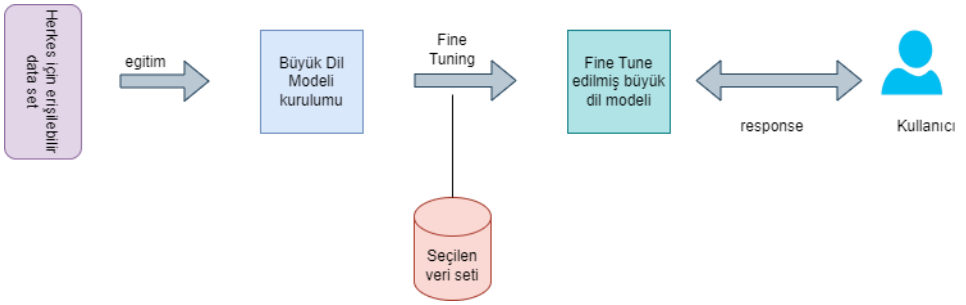
Büyük dil modellerinin belirli bir alandaki daha küçük veriler üzerinde eğitmektedir. Eğitim sonucu belirli bir görevde daha iyi performans vermesini sağlamak için ince ayar vermektir.



Fine Tune Aşamaları:

Herkese açık verilerden model eğitilir. Bu model ön eğitimli modeldir. Ön eğitimli modelleri geliştirmek yüksek bilgisayar gücü gerektirmektedir. Maliyetten dolayı büyük dil modellerinin Google, Meta, OpenAI büyük teknoloji şirketleri eğitir ve biz bu ön eğitimli modeli alıp kendi veri setimizle fine tune ederiz. Farklı teknikler kullanarak modelin bütün parametrelerini eğitmek yerine parametrelerin bir kısmını dondurup bir kısmını da eğiteceğiz. Bu teknikleri hugging face içindeki PEFT kütüphanesiyle gerçekleştireceğiz.

Ek olarak büyük dil modeliyle çalışacağımız için Quantization ile modelin boyutunu azaltma işlemi gerçekleştirilecektir.



Şekil 4: Fine Tuning Aşamaları

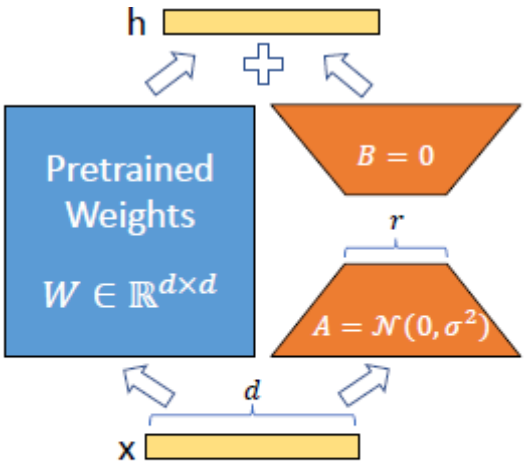


Figure 1: Our reparametrization. We only train A and B.

Model ağırlıklarının sadece bir kısmı dondurulur, kalan küçük bir kısmı veri setine göre eğitilir. Bu sayede zaman ve maliyet tasarrufu sağlanır.

7. GERÇEKLEŞTİRİM

Quantization ile Modeli Yükleme

```
[ ]: !pip install -q -U transformers bitsandbytes peft datasets accelerate trl
```

Şekil 1 - Kütüphaneler yüklenir

Şekil 1'de kullanacağımız; transformers, bitsandbytes, peft , veri setimizi yüklemek için dataset ve modeli eğitmek için trl kütüphanelerini pip metodu ile yükleriz. -q ekrana çok fazla kod yazılmaması için kullanılırken -U varolan kütüphaneleri güncellemek için kullanılır.

```
[ ]:
from transformers import AutoTokenizer

base_model = "mistralai/Mistral-7B-v0.1"

tokenizer = AutoTokenizer.from_pretrained(
    base_model,
    padding_side = "right",
    add_eos_token = True,
)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.add_bos_token, tokenizer.add_eos_token
```

Şekil 2 - Tokenizer yükleme

Şekil 2'de base model değişkeni oluşturularak mistral modelimiz bu değişkene atanır. Sonrasında bu değişken tokenizer sınıfına verilir. AutoTokenizer sınıfından from_pretrained metodu ile oluşturulan base model değişkeni, maksimum dizi uzunluğundan az olan cümleleri metnin sağ tarafından doldurmak için padding tokenları padding_side right olarak ayarlanır. Ardından dizinin bittiğini gösteren eos tokenının dizi sonuna eklenmesi için add_eos_token parametresi True olarak ayarlanır. Tokenizerlar cümlenin bittiğini modele söylecek şekilde ayarlanır. Böylece Tokenizer'ı yüklemiş olduk. Son olarak dizinin başladığını ve bittiğini gösteren tokenlarımız (bos_token ve eos_token) yüklenmiş mi kontrol ettik.

```
[ ]:
import torch
from transformers import BitsAndBytesConfig

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=False,
    bnb_4bit_compute_dtype=torch.bfloat16
)
```

Şekil 3 - Model ağırlıkları düzenlenir

Şekil 3'de Transformers kütüphanesinden BitsAndBytes sınıfı import edilerek modelin ağırlıkları ayarlanır. Bnb_config değişkeni oluştuşturulur ve BitsAndBytes sınıfı çağrılarak load_in_4bit parametresi True yapılır. Bnb_4bit_quant_type ile veri tipimizi nf4 olarak

ayarlarız. Bnb_4bit_use_double_quant parametresi ikinci bir quantization kullanılır fakat biz modelimizin parametrelerini daha fazla küçültmek istemediğimiz için bu parametreyi False değerini verdik. En son olarak torch kütüphanesi yüklenerek torch içindeki bfloat16 ayarlanır böylece ağırlık hesaplamaları 16 bitlik float veri tipinde yapabildik. Böylece bitsandbytes ayarlamaları yapıldı ve configurasyon değişkenimiz oluşmuş oldu.

```
from transformers import AutoModelForCausalLM

model = AutoModelForCausalLM.from_pretrained(
    base_model,
    load_in_4bit=True,
    quantization_config=bnb_config,
    torch_dtype=torch.bfloat16,
    device_map="auto",
)
```

Şekil 4 - Model yüklenir

Şekil 4'de modeli yükledik. Öncelikle transformers kütüphanesinden AutoModelForCausalLM sınıfı yüklenir. Model değişkeni oluşturulur from_pretrained metodu çağrıldı. Sonrasında modeli 4 bit olarak yükleneceği için parametre True yapıldı. Quantization_config parametresine öncesinde oluşturduğumuz bng_config değişkeni verildi. İşlem yapılacak veri tipi ayarlandı. CPU'yu modelin otomatik olarak kullanması için device_map parametresi auto yapıldı.

Veri Seti Yükleme

```
from datasets import load_dataset

dataset_name = "databricks/databricks-dolly-15k"

train_dataset = load_dataset(dataset_name, split="train[0:800]")
eval_dataset = load_dataset(dataset_name, split="train[800:1000]")
```

Şekil 5 - Verseti Yüklenir

Şekil 5'de Hugging Face'den aldığımız DataBricks tarafından geliştirilen dolly-15k veri setimizi yükledik. Veri setimizde 15 bin örneklem bulunmaktadır. Datasets kütüphanesinden

load_datset metordu import edilir. Verisetimizin yolu dataset_name değişkenine atanır. Eğitim verisetimizin 0'dan 800'e kadar olan verileri değerlendirme verisetine ise 800. satıdan 1000. satıra kadar olan verileri içerir.

Veri setimizde 4 tane sütun ve 800 satır bulunmaktadır.

```
train_dataset
```

```
train_dataset.to_pandas()
```

```
train_dataset.to_pandas().dtypes
```

```
train_dataset.to_pandas().value_counts("category")
```

Şekil 6 - Veriseti pandas formatına çevrilir

Şekil 6'da to_pandas metodu ile tablo yapısına getirildi. Veri setimizde kullanacağımız dtypes özniteliği çağrılır sonrasında kategori sütunundaki kategori sayıları kontrol edildi.

```
def generate_prompt(sample):
    full_prompt =f"""<s>[INST]{sample['instruction']}
    {f"Here is some context: {sample['context']}" if len(sample["context"]) > 0 else None}
    [/INST] {sample['response']}</s>"""
    return {"text": full_prompt}
```

Şekil 7 - Prompt Formatı Ayarlanır

Şekil 7'de generate_prompt fonsiyonuna sample isminde bir parametre alıyor ve bu sample içindeki sütunları ilgili yerlere yerleştirecek sample veri setimiz oluyor veri setindeki sütunları kullanmak için f string kullandık. Full_prompt isminde bir değişken oluşturularak prompt'un başında gelecek tokenlar ve dizinin bittiği s isminde özel tokenlar yazılır. Eğer bir context varsa if string içindeki alana yazılır. Çünkü veri setinde her bir satırda context yok bu yüzden if deyimi ile yazıldı. Eğer context uzunluğu sıfırdan büyükse metin yazılsın şeklinde

ayarladık. Sonrasında veri setindeki response sütunu kullanılarak metnin cevap kısmı yazılır. Sonrasında full_prompt değişkenini return ile döndürdük.

```
train_dataset[0]
```

```
print(generate_prompt(train_dataset[0]))
```

```
generated_train_dataset = train_dataset.map(
    generate_prompt, remove_columns=list(train_dataset.features))
generated_val_dataset = eval_dataset.map(
    generate_prompt, remove_columns=list(train_dataset.features))
```

Şekil 8 - Propmt Kontrolleri

Şekil 8'de öncelikle veri setimizin ilk satırını ekrana yazdırdık (train_dataset[0]. Sonrasında bu veriyi generate_prompt fonksiyonumuza girdik ve ekrana yazdırdık. Metnin başladığını ve bittiğini gösteren özel tokenlarımızı ekranda kontrol etmiş olduk.

Sonrasında generate_train_dataset değişkenine map metodu ile propmt üretme fonksiyonu girilir ve metinde sadece text verisi olması için ayarlanır. Sonrasında aynı işlem değerlendirme veri setimiz içinde uyguladık. Böylece generate_propmt fonskiyonumuz tüm verilere uygulanmış oldu.

```
generated_train_dataset
```

```
generated_train_dataset[5]["text"]
```

Şekil 9 - Veri Formatları Kontrol Edilir

Şekil 9'da verilerin formatları ayarlanıp ayarlanmadığını kontrol edildi. Örneğin eğitim veri

setinin 5. indexdeki satırını ekrana yazdırdık. Veri setimizde sadece text sütunun kaldığını kontrol edildi.

PEFT ile Model Konfigürasyonu

```
from peft import prepare_model_for_kbit_training

model.gradient_checkpointing_enable()

model = prepare_model_for_kbit_training(model)
```

Şekil 10 - kbit ile Model Eğitimi

Şekil 10'da öncelikle peft kütüphanesinden prepare_model_for_kbit_training metodu yüklendi. Sonrasında model için gradient kontrol noktası oluşturuldu. Model.gradient_checkpointing_enable() satırında, modelimiz büyük olduğu için sadece stratejik ağırlıkları eklenir böylece bellekte çok fazla veri saklanmaz. Ardından öncesinde import ettiğimiz prepare_model_for_kbit_training metodunun içine modelimizi vererek ile modeli kbit eğitim için hazırladık.

```
def print_trainable_parameters(model):
    trainable_params = 0
    all_param = 0
    for _, param in model.named_parameters():
        all_param += param.numel()
        if param.requires_grad:
            trainable_params += param.numel()
    print(
        f"trainable params: {trainable_params} || all params: {all_param} || trainable%: {100 * trainable_params / all_param}"
    )
```

Şekil 11- Modelde Eğitilecek Parametre Sayıları

Şekil 11'de print_trainable_parameters fonsiyonu kullanarak tek tek eğitilecek parametreleri hesaplamak için bir for döngüsü oluşturulur ve öncelikle tüm parametreler hesaplatılır ardından eğitilecek parametreleri bulmak için if deyimi kullanılır. For döngüsü ile tüm parametreler ve eğitilecek parametreler yazdırmış olduk. Bu değerler ekrana print fonksiyonu ile eğitilebilecek parametrelerin yüzdesi yazdırılır. Modelde eğitilecek parametrelerin sayısını hesaplayan fonksiyonumuzu yazmış olduk.

```python
from peft import LoraConfig, get_peft_model

lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=[
        "q_proj",
        "k_proj",
        "v_proj",
        "o_proj",
        "gate_proj",
        "up_proj",
        "down_proj",
        "lm_head",
    ],
    bias="none",
    lora_dropout=0.05,
    task_type="CAUSAL_LM",
)
```

Şekil 12 - Lora Konfigurasyonu için Kullanılacak Metotların İmport Edilmesi

Şekil 12'de Lora konfigürasyonu için kullanılacak metotlar yüklenir. Öncelikle peft kütüphanesinden LoraConfig ve get_peft_model metotları import edildi. Burada eğitilecek parametrelerin sayısı ayarlandı. Lora_alpha parametresi öğrenilecek ağırlıklar için ölçekleme parametresidir bu parametreyi 16 ayarladık. Böylece ince ayarları verilere öncelik vererek hesaplama komplesini azaltmış olduk. Sonrasında target için modüller ayarlandı bu modüller lora uygulanacak lineer katmanlar. Sonrasında Lora için bias parametresi none girildi. Dropout parametresi ile katmanlar arasında 0.05 lik bir regülerleşme kullanıldı. Görevimiz metin üretme bir diziyi tamamlama olduğu için tast_type parametresi ayarlandı.

```
from peft import get_peft_model

model = get_peft_model(model, lora_config)

print_trainable_parameters(model)
```

```
print(model)
```

Şekil 13 - Lora Konfigurasyonunun Modele Uygulanması

Şekil 13'de get_peft_model metodu peft kütüphanesinden yüklenerek modelimi ve lora_config değişkenimizi yazdık. Böylece Lora tekniğiyle eğitim yapacağımızı modelimize söylemiş olduk. Daha sonrasında print_trainable_parameters ile eğitilecek parametre sayısı ve tüm parametre sayısı ekrana gelir. Eğitilecek parametre sayısı tüm parametrelerin 0.56 lık kısmıdır. Print fonksiyonu ile modele eklenen Lora katmanları gözlemlendi. Böylece peft mimarisi ve uygulanacak katmanları ve ayarlar görülmüş oldu.

Model Eğitimi

```
from transformers import TrainingArguments

training_arguments = TrainingArguments(
    output_dir="./results",
    num_train_epochs=1,
    per_device_train_batch_size=4,
    gradient_accumulation_steps=1,
    optim="paged_adamw_32bit",
    save_strategy="steps",
    save_steps=25,
    logging_steps=25,
    learning_rate=2e-4,
    weight_decay=0.001,
    max_steps=50,
    evaluation_strategy="steps",
    eval_steps=25,
    do_eval=True,
    report_to="none",
)
```

Şekil 14 - Eğitilen Argümanların Ayarlanması

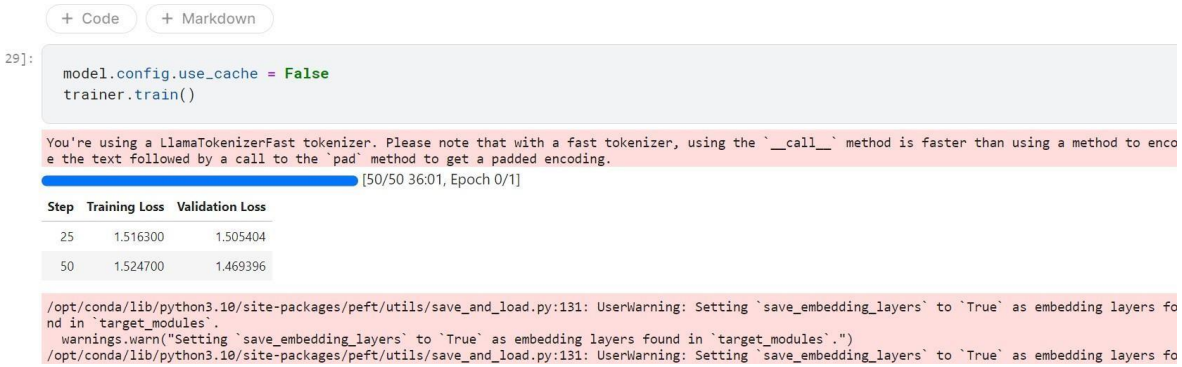
Şekil 14'de transformers kütüphasinden TrainingArguments sınıfı import edilir. Bu sınıf kullanarak modeli kaydedeceğimiz dizin olarak result ismindeki dosyamızın altına kaydettik. Epochs sayısına 1 değeri verildi. Gradientleri küçük artışlarla hesaplamak için parametreye bir verilerel eğitim esnasında gradientleri biriktirerek ve graidentların daha küçük gruplar halinde iteratif olarak hesaplanmasını sağlar. Sonrasında model kaydetme stratejisi steps olarak kaydedilir. Her 25 adımda bir checkpointleri kaydettik. Öğrenme oranını learning_rate belirlendi. Max_step parametresi eğitim için maximum adım sayısı 50 olarak ayarlandı. Eval_steps parametresi 25 olarak belirlenerek checkpointleri kaydetme adımları belirlendi. Eğitim bittikten sonra modelin değerlendirilmesi için do_eval parametresi True olarak ayarlandı.Raporlama yapılamaması için report_to parametresi none olarak ayarlandı.

```python
from trl import SFTTrainer

# Setting sft parameters
trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    args=training_arguments,
    train_dataset=generated_train_dataset,
    eval_dataset=generated_val_dataset,
    peft_config=lora_config,
    dataset_text_field="text",
)
```

Şekil 15 - SFTTrainer Sınıfı ile Model Eğitimi

Şekil 15'te SFFTTrainer sınıfı ile modelimizi eğittik. Hugging Face içindeki trl kütüphanesinden SFTTrainer sınıfını yükledik. Bu sınıftan trainer objesini oluşturduk. Trainer objesi içine modelimizi, tokenizer parametresini ve daha önce oluşturduğumuz argümanları args parametresine girdik, train_datsets parametresine eğitim veri setimizi yazdık. Peft_config parametresine lora konfigürasyonumuzu yazdık. En son olarak verisetimizdeki text sütunumuzu dataset_text_field parametresine verdik.



Şekil 16 - Model Eğitimi

Şekil 16'da eğitim aşamasında olduğumuz için model config içerisindeki use_cache ayarını False olarak ayarladık. Daha sonrasında train metodunu çağırarak modelimizi eğittik.

```
my_finetuned_model = "gamzeyy/mistral-7b-dolly"

trainer.model.push_to_hub(my_finetuned_model)
```

Şekil 17 - Fine Tune Edilen Modelin Hub'a Yüklenmesi

Şekil 17'de my_finetuned_model değişkeni oluşturularak huba yüklemek için modelimizin ismi yazıldı. Ardından trainer'dan model çağrıldı sonrasında push_to_hub metodu ile değişkenimizin ismini yazdık. Eğittiğimiz modeli bu şekilde hub'a gönderdik.

PEFT Model ile Metin Üretme

Bu aşamada Lora ile eğittiğimiz ağırlıkları indirerek ve bu ağırlıkları orijinal modelle birleştirdik. Daha sonra bu modeli çıkarım yapmak için kullanıldı. Fine tune ettiğimiz modeli öncelikle yükledik sonrasında yeni tahminlerde kullandık.

```
!pip install transformers
```

Şekil 18 - Tranformers Kütüphanesinin yüklenmesi

Şekil 18’de Transformers kütüphanesi yüklendi.

```
!pip install gradio
```

Şekil 19 - Gradio'nun Yüklenmesi

Şekil 19’da projemize arayüz yapabilmek için gradio yüklendi.

```
!pip install -q -U peft bitsandbytes
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 168.3/168.3 kB 4.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 105.0/105.0 MB 8.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 270.9/270.9 kB 30.5 MB/s eta 0:00:00
```

Şekil 20 - Gerekli Kütüphanelerin Yüklenmesi

Şekil 20’de Peft ve BitsAndBytes kütüphaneleri yüklendi.

```
import torch
from peft import PeftModel, PeftConfig
from transformers import AutoModelForCausalLM, AutoTokenizer
```

Şekil 21 - Gerekli Diğer Kütüphaneler import edilir

Şekil 21’de torch kütüphanesi import edildi. Sonrasında PeftModel ve PeftConfig sınıfı import edildi. Transformers kütüphanesinden AutoModelForCausalLM ve AutoTokenizer sınıfları import edildi.

```
peft_model_id = "gamzeyy/mistral-7b-dolly"

config = PeftConfig.from_pretrained(peft_model_id)
```

Şekil 22 - Peft Model Configurasyonu

Şekil 22’de Peft_model_id değişkenine modelimizin yolu verilir. Sonrasında peft modelin konfigürasyonu config değişkenine atanır.

```
from transformers import BitsAndBytesConfig
```

Şekil 23- BitsAndBytes Kütüphanesinin Yüklenmesi

Şekil 23'de BitsANsBytesConfig sınıfı Transformers kütüphenesinden import edilir.

```
model = AutoModelForCausalLM.from_pretrained(
    config.base_model_name_or_path,
    return_dict=True,
    load_in_4bit=True, #agırlıkları 4 bit


)
```

Şekil 24 - Modelin Yüklenmesi

Şekil 23'de AutoModelForCausalLM sınıfındaki from_pretrained metodu ile modelimiz yükledik. Öncelikle base modelimizi config içerisinden aldık. Çıktıların sözlü yapısında dönmesi için return_dick parametresini True olarak ayarladık. Ağırlıkları 4 bit olarak yüklemek için load_in_4bit parametresi True olarak ayarlandı.

```
#Tokenezier yükleme
tokenizer = AutoTokenizer.from_pretrained(config.base_model_name_or_path,
                                          padding_side = "right", #sagdaki boşluk doldurmak için
                                          add_eos_token = True)   #dizinin sonuna metin bittignii gösteren özel token eklenir
tokenizer.pad_token = tokenizer.eos_token
```

Şekil 25 - Tokenizerın Yüklenmesi

Şekil 25'de verileri modele vermeden önce ön işleme yapmak için AutoTokenizer sınıfındaki from_pretrained metodu ile tokenizer yüklendi. Sağ taraftaki boşlukları doldurmak için padding_side parametresine right yazıldı. Dizinin sonuna metnin bittiğini göstermek için özel token ekledik (add_eos_token). Ardından pad tokena tokenizer içindeki eos token'ı yazıldı. Bu şekilde tokenizer çalıştırıldı.

```
fine_tuned_model = PeftModel.from_pretrained(model, peft_model_id) #basemodel + egitilen modeli birleştir
```

Şekil 26 - Base Model ile Peft Modelin Birleştirilmesi

Şekil 26'da Base modelile Peft modeli birleştirdik. Bunun için fine_tuned_model isminde bir değişken aldık ve peft model sınıfından from_pretrained metodu ile öncelikle modelimzii sonrasında peft modelimizi yazdık. Bu şekilde orijinal modelle eğittiğimiz modelin ağırlıklarını birleştirmiş olduk.

```
from transformers import pipeline, logging

logging.set_verbosity(logging.CRITICAL)

pipe = pipeline(
    task="text-generation",
    model=fine_tuned_model,
    tokenizer=tokenizer,
    eos_token_id=model.config.eos_token_id,
    max_new_tokens=100)
```

Şekil 27 – Pipeline ile Çıkarım Yapılması

Şekil 27'de pipeline ve sadece önemli uyarıların ekrana yazılması için logging metodunu import ettik. Öncelikle sadece kritik uyarıları görmek için logging içindeki set_verbosity metodunu kullandık. Sonrasında pipe değişkeni oluşturduk ve pipeline ile çıkarım yapmak için öncelikle task'ımzıı yazdık, model parametresine fine tüne ettiğimiz modeli yazdık, tokenizer parametresine tokenizer'ımızı verdik, dizinin bittiğini gösteren tokenizi ayarlardık ve son olarak üretilecek maximum token sayısını 100 olarak ayarladık.

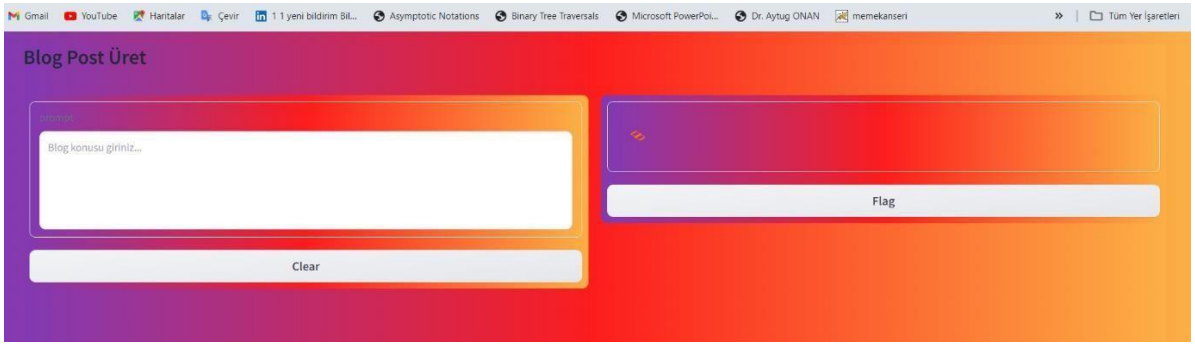
```
prompt = """
What is a Python?  Here is some context: Python is a high-level, general-purpose programming language.
"""
pipe = pipeline(task="text-generation",
                model=fine_tuned_model,
                tokenizer=tokenizer,
                eos_token_id=model.config.eos_token_id,
                max_new_tokens=100)

result = pipe(f"<s>[INST] {prompt} [/INST]") #prompt için özel tokenler
generated = result[0]['generated_text']
print(generated[generated.find('[/INST]')+8:])
```

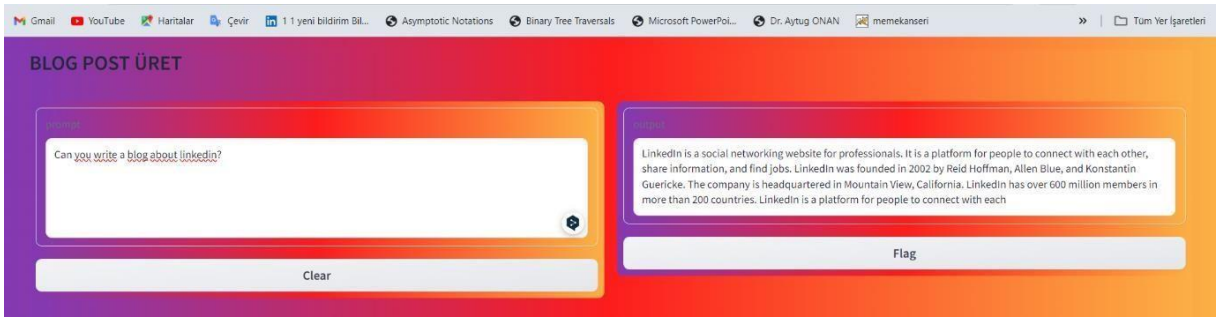
Şekil 28 - Prompt Oluşturulması

Şekil 28'de önceden oluşturduğumzu prompt formatı kullanıldı. Bu soru için bir içerik girildi. Sonrasında bu prompt pipeline'a verildi. Prompt için özel tokenlerımız yazıldı.Yapılan tahmini ekrana yazdırmak için generate_text değişkeni alındı. Print ile generated değişkeni içindeki find metonu kullanarak 8. karekterden sonraki değerleri ekrana verildi. Böyle fine tune ettiğimiz model ile blog post metin üretmiş olduk.

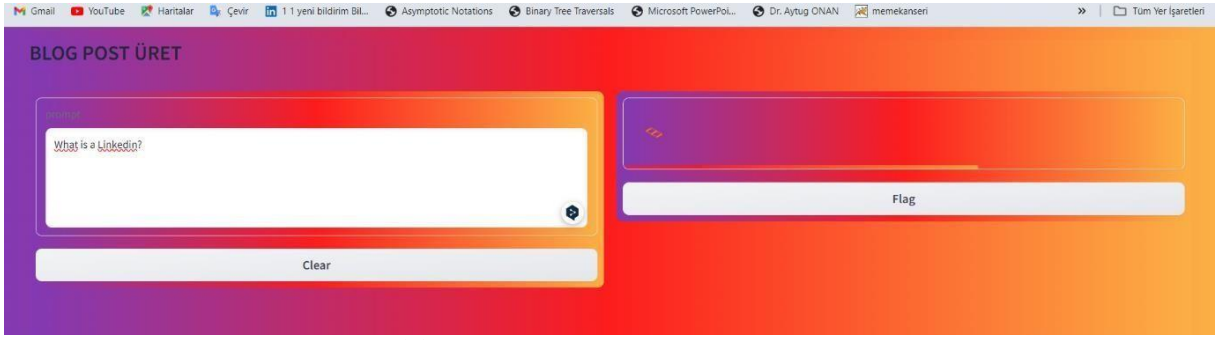
ARAYÜZLER



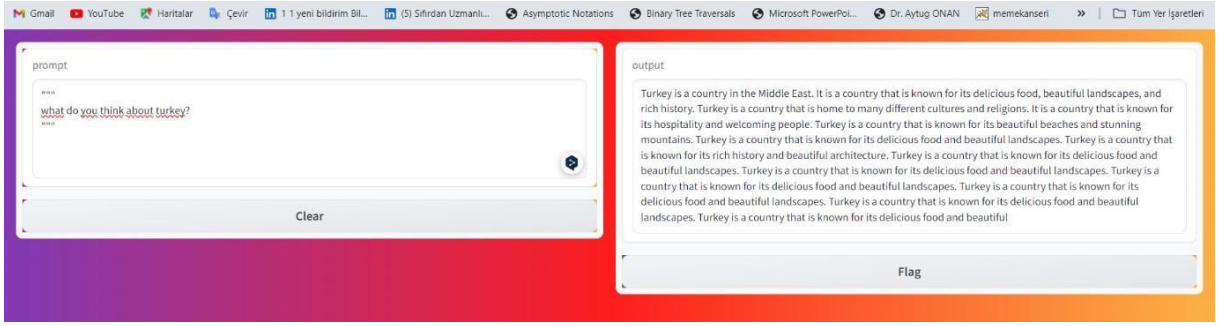
Şekil 29 - Arayüz Ana Ekran



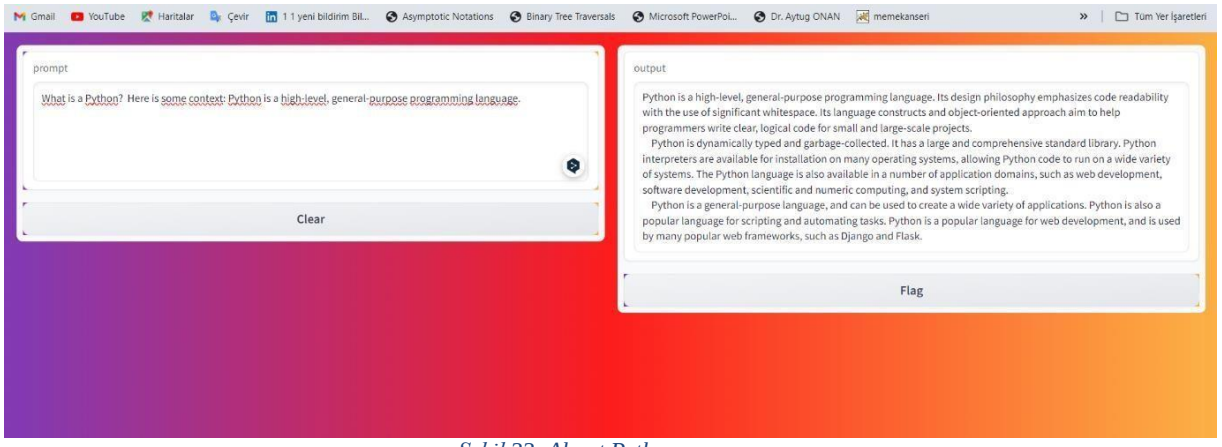
Şekil 30 - Can you write a blog about linkedin ?



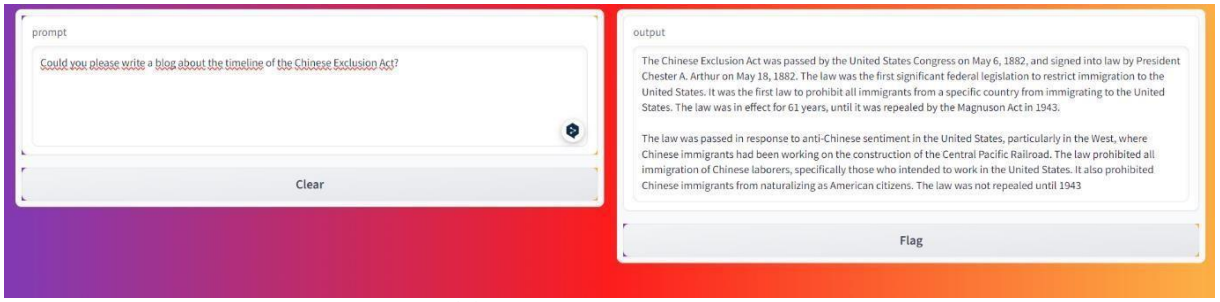
Şekil 31 - What is linkedin blog post üretiliyor



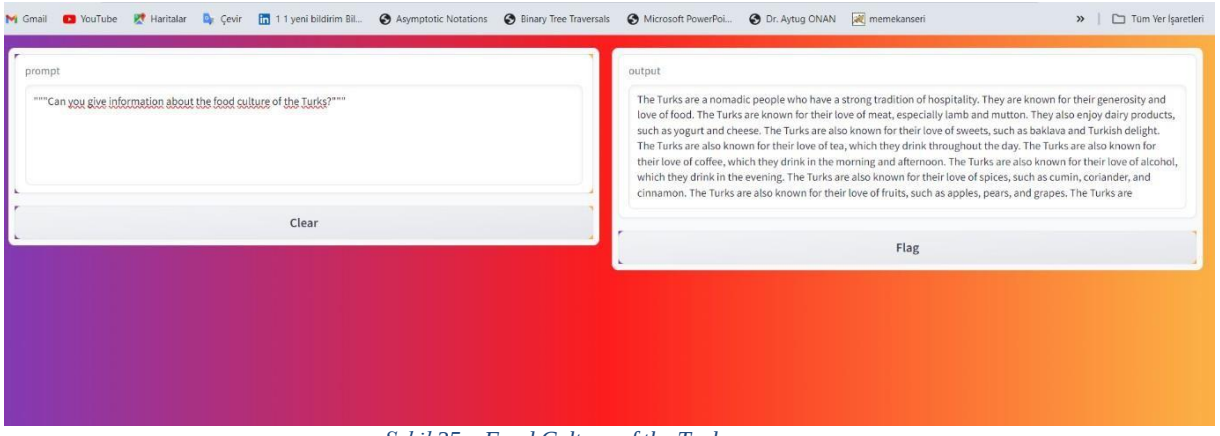
Şekil 32 – About Turkey



Şekil 33- About Python



Şekil 34 – About Chinese Exclusion Act



Şekil 35 – Food Culture of the Turks

8. SONUÇ VE DEĞERLENDİRME

Projemizde NLP alanındaki en modern teknikleri kullanarak büyük dil modeli olan Mistral modeli LORA tekniği ile fine tune ettik. Projemize Kaggle Notebook üzerinde çalıştık.

Öncelikle kullanacağımız kütüphaneleri yükledik. Toplamda 6 tane kütüphane kullandık. Ardından veri ön işleme için kullanacağımız Mistral modelin tokenizerını yükledik. Bu tokenizerı yüklerken AutoTokenizer sınıfını kullandık. Hagging face içindeki tokenizer kullanılarak metinler sayısal verilere dönüştürülür. Ardından model yükleme aşamasına geçtik. Modelimiz 7 milyar parametresi olan Mistral 7b modelidir. Modelimiz ile çalışmak için büyük bir 28 GB'lık RAM gerektirdiği için modelimizin ağırlıklarını 4 bite indirerek model boyutunu küçülttük. Bu işlemi BitsAndBytesConfig sınıfı ile yaptık. Bu sınıfı import ettik ve bu sınıftan bir obje oluşturduk. Sonrasında bir metinden sonraki tokenları tahmin edebilmek için transformers kütüphanesinden AutoModelForCausalLM sınıfını import ettik. Bu sınıfa modeli 4 bit olarak yükleyeceğimizi söyledik ve quantization_config parametresine oluşturduğumuz konfigürasyon değişkenini atadık veri tipi olarak da torch kütüphanesi içerisindeki bfloat16 veri tipini kullandık. Bu şekilde boyutunu küçülttüğümüz modeli yükledik.

Modeli fine tune etmek için kullanacağımız hugging face'den aldığımız veri setimizi yükledik. Verisetimiz DataBricks-dolly-15k veriseti olup içerisinde 15 bin örneklem barındırmaktadır. Verisetini yükledikten sonra verileri anlamak için pandas'a çevirdik. Pandas'a çevirmek için to_pandas metodunu kullandık. Sonrasında verileri promp'a dönüştürmek için generate_prompt fonskiyonunu oluşturduk. Bu fonksiyona başlangıç ve bitiş özel tokenlarını ekledik. Prompt formatını ayarlamak için oluşturduğumuz fonksiyonu bütün verisetlerine (eğitim ve değerlendirme verisetleri) uyguladık.

Modeli kurarken LoRA tekniğini kullandık. Bu teknikle modelin çok az bir ağırlığını eğittik. Öncelikle LoRA için bir konfigürasyon dosyası oluşturduk. Ardından kaç tane parametre eğiteceğimizi ekrana yazdırdık. Eğitim parametrelerimizi ayarladık ve SFTTrainer sınıfına modeli, tokenizer'ı , eğitim argümanlarını, eğitim ve test verilerini , lora konfigürasyonunu ve metnin bulunduğu anahtarı verdik. Böylece trainer objemizi oluşturduk. Sonrasında train metodunu çağırarak modelimizi eğittik. Eğittiğimiz modeli

push_to_hub metodu ile hub'a gönderdik.

Fine tune ettiğimiz ağırlıkları kullanarak tahminlerimizi yapmaya başladık. Tahmin yapabilmek için peft modelimizi hub'dan indirdik. Daha sonra orijinal modelimizi hub'dan indirdik. Tokenizer'ımızı yükledik. Ardından orijinal model ile peft modeli birleştirdik. Bu şekilde fine tune ettiğimiz modelimizi oluşturmuş olduk. Bu modeli kullanarak tahminler yapabilmek için bir pipeline kullandık. Blog post metin oluşturabilmek için pipeline'a prompt ile giriyoruz ve böylece fine tune ettiğimiz modele göre blog post metinleri üretmiş olduk.

KAYNAKÇA

[1] Korkmaz, T. 1996. Turkish Text Generation with Systemic-Functional Grammar. Master's Thesis, Bilkent University, Department of Computer Engineering and Information Science, Ankara.

[2] Gündoğdu, Ö.E. ve Duru, N. 2016. Türkçe Metin Özetlemede Kullanılan Yöntemler. 18. Akademik Bilişim Konferansı, Adnan Menderes Üniversitesi, 30 Ocak-5 Şubat, Aydın

[3] Kazkılınç, S. 2013. Türkçe Metinlerin Etiketlenmesi. Yüksek Lisans Tezi, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

[4] Brown, P.F. et al. 1990. Class-Based N-Gram Models of Natural Language. In Proceedings of the IBM Natural Language ITL, Paris, pp. 283–298.

[5] Mairesse, F. et al. 2010. Phrase-based Statistical Language Generation using Graphical Models and Active Learning. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL), Uppsala.

[6] Uchimoto, K. et al. 2002. Text Generation from Keywords. COLING '02 Proceedings of the 19th International Conference on Computational Linguistics, Taipei.

[7] Tan, J. et al. 2017. From neural sentence summarization to headline generation: a coarse-tofine approach. 26th International Joint Conference on Artificial Intelligence (IJCAI-17), 19-25 August, Melbourne-Australia, p.4109-4115.

[8] Bauer, A. et al. 2015. Rule-based Approach to Text Generation in Natural Language-Automated Text Markup Language.

[9] Fagroud, Fatima Zahra and Rachdi, Muhammed and Lahmar, El Habib Ben. 2022. Automatic Story Generation: Case Study of English Children's Story Generation Using GPT-2

[10] Kutlugün, Mehmet Ali, and Şirin, Yahya. 2018. Turkish meaningful text generation with class based n-gram model.

[11] Özdemir, C.B. ve Amasyalı, M.F. 2010. Hayat Bilgisi Veritabanı Kullanılarak Otomatik Cümle Üretimi. XV. Türkiye'de İnternet Konferansı, 2-4 Aralık, İstanbul, cilt.1 s.1-4.

[12] Adalı, Ş. ve Erenler, Y. 2003. Türkçe için Okuma Fonksiyonlu Otomatik Metin Oluşturma Sistemi. Elektrik-Elektronik-Bilgisayar Mühendisliği 10. Ulusal Kongresi, İstanbul Sayfa:484-487.

Medium, "Text Generation with Hugging Face Transformers: A Beginner's Guide", erişim: 5 Şubat 2023, https://medium.com/@lokaregns/text-generation-with-hugging-face-transformers-a-beginners-guide-6b0b4b957379

Datacamp,"What is Text Generation", erişim : 2021, https://www.datacamp.com/blog/what-is-text-generation