

## Web Trafik Loglarına Dayalı Yapay Zeka Destekli Soru-Cevap Sistemi Geliştirme

### 1. Aşama:

Öncelikle web trafik loglarını oluşturmam için Windows kullanıcısı olarak XAMPP'i indirdim. XAMPP, Apache, MySQL, PHP, Perl gibi bileşenleri sunan bir yazılım paketidir.

Bu yöntemle Apache web sunucusu kurabildim. Sunucumda trafik oluşturmak için tarayıcı üzerinden istekler yaptım. Apache sunucusu, bu istekleri log dosyasına kaydetti.

Log kayıtlarının bulunduğu dosyamın yolu C:\xampp\apache\logs\access.log 'dir.

Örnek olarak log dosyamdan bir satır:

```
:::1 - - [12/Aug/2024:09:31:46 +0300] "GET / HTTP/1.1" 302 - "-"  
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,  
like Gecko) Chrome/127.0.0.0 Safari/537.36"
```

- :::1: İstemcinin IP adresi (Bu durumda :::1, yerel makineden gelen bir IPv6 adresidir).
- - -: Kullanıcı kimliği ve kimlik doğrulaması (genellikle boş bırakılır).
- [12/Aug/2024:09:31:46 +0300]: İsteğin zaman damgası (tarih ve saat).
- "GET / HTTP/1.1": HTTP isteğinin türü, yolu ve HTTP sürümü (bu örnekte bir GET isteği).
- 302: HTTP durum kodu (bu örnekte, 302 bir yönlendirme olduğunu gösterir).
- -: İstekle ilişkili bayt boyutu (bu örnekte yok).
- "-": İsteği yapan sayfanın referansı (bu örnekte boş).
- "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36": İstemcinin user-agent bilgisi, yani tarayıcı ve işletim sistemi bilgisi.

Projede kullandığım kütüphaneler ve modüller:

```
import re  
import pandas as pd  
from sklearn.feature_extraction.text import TfidfVectorizer  
import faiss  
import numpy as np  
from transformers import GPT2LMHeadModel, GPT2Tokenizer
```

- **re:** Düzenli ifadelerle metin işlemleri için kullanılır.
- **pandas:** Veri analizi ve işleme için kullanılır.
- **sklearn.feature\_extraction.text.TfidfVectorizer:** TF-IDF vektörizasyonu için kullanılır.
- **faiss:** Vektörleri hızlı bir şekilde indeksleyip aramak için kullanılır.
- **numpy:** Nümerik işlemler ve diziler için kullanılır.
- **transformers:** Hugging Face'in GPT-2 modelini ve tokenizer'ını yüklemek için kullanılır.

## Log dosyasını okuma:

```
log_file_path = 'access.log'

with open(log_file_path, 'r') as file:
    logs = file.readlines()
```

- `log_file_path`: Log dosyasının yolu.
- `logs`: Log dosyasındaki tüm satırları okur ve bir liste olarak saklar.

## Log satırlarını ayrıştırma:

```
log_pattern = re.compile(
    r'(?P<ip>\d+\.\d+\.\d+\.\d+)' # IP adresi
    r' - - ' # Sabit metin
    r'\[(?P<timestamp>[^\]]+)\]' # Zaman damgası
    r'"(?P<method>\w+)' # HTTP methodu
    r'(?P<url>[^\s]+)' # URL
    r' [^"]+"' # Protokol ve diğer sabit metinler
)

parsed_logs = []

for log in logs:
    match = log_pattern.match(log)
    if match:
        parsed_logs.append(match.groupdict())
```

- `log_pattern`: Log satırlarını ayrıştırmak için kullanılan düzenli ifade deseni.
- `parsed_logs`: Ayrıştırılan logları saklamak için kullanılan liste.
- `match.groupdict()`: Ayrıştırılan log verilerini sözlük olarak döndürür.

## Verileri DataFrame'e Dönüştürme:

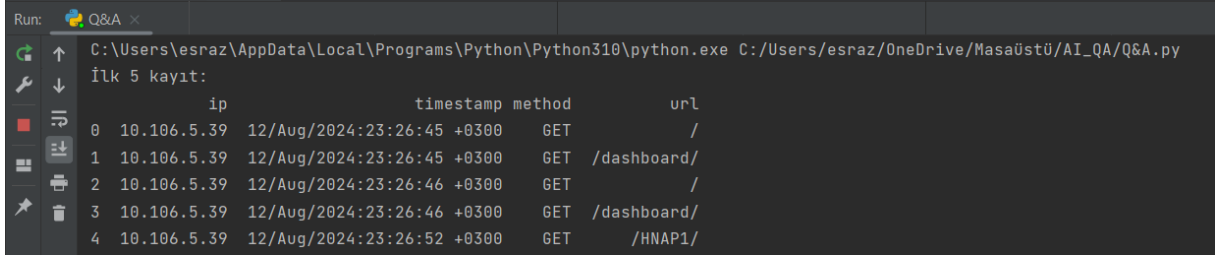
```
df = pd.DataFrame(parsed_logs)

if df.empty:
    print("Log verileri düzgün ayrıştırılamadı. Lütfen log dosyasını
    ve düzenli ifade desenini kontrol edin.")
    exit(1)

print("İlk 5 kayıt:")
print(df.head())
```

- `df`: Ayrıştırılmış logları içeren DataFrame.
- `df.empty`: DataFrame'in boş olup olmadığını kontrol eder.

Ödevde istenildiği üzere log dosyasındaki gerekli verileri seçme, temizleme ve yapılandırma işlemini (örneğin, IP adresleri, erişilen sayfalar, zaman damgaları gibi verileri ayıklama) yaptıktan sonra ayrıştırılmış log verilerini DataFrame'e dönüştürdüm ve bu DataFrame'e ait ilk 5 kayıt:



```
Run: Q&A x
C:\Users\esraz\AppData\Local\Programs\Python\Python310\python.exe C:/Users/esraz/OneDrive/Masaüstü/AI_QA/Q&A.py
İlk 5 kayıt:
   ip          timestamp method url
0  10.106.5.39  12/Aug/2024:23:26:45 +0300 GET /
1  10.106.5.39  12/Aug/2024:23:26:45 +0300 GET /dashboard/
2  10.106.5.39  12/Aug/2024:23:26:46 +0300 GET /
3  10.106.5.39  12/Aug/2024:23:26:46 +0300 GET /dashboard/
4  10.106.5.39  12/Aug/2024:23:26:52 +0300 GET /HNAP1/
```

## **TF-IDF Vektörizasyonu:**

TF-IDF, bir dokümandaki herhangi bir kelimenin dokümanla ne kadar alakalı olduğunu gösteren bir sayısal ölçümdür. TF-IDF kelimelerin birbirlerinden bağımsız olduğunu kabul etmekte ve kelimeler arasındaki anlamsal ilişkiyi ifade edememektedir. [1]

### **Bu Projede TF-IDF Vektörizasyonunun Kullanımı ve Amacı:**

Bu projede, Apache log dosyalarındaki kayıtlar (özellikle URL'ler) metin verisi olarak ele alındı. Her bir log kaydının içeriği, TF-IDF vektörleri oluşturmak için kullanıldı. İşlem şu adımlarla gerçekleştirildi:

#### **1. Log Verilerinin Birleştirilmesi:**

- Her bir log kaydı, IP adresi, zaman damgası, HTTP yöntemi ve URL gibi bilgilerin bir araya getirilmesiyle tek bir metin satırı olarak oluşturuldu. Bu satırlar, vektörizasyon işlemine tabi tutuldu.

#### **2. TF-IDF Vektörizasyonu:**

- Bu birleştirilmiş metin satırları, TF-IDF Vektörizer aracılığıyla dönüştürüldü. Bu, her bir log kaydının, kelime frekanslarını ve bu kelimelerin diğer loglarda ne kadar nadir bulunduğunu dikkate alan bir vektör ile temsil edilmesini sağladı.

#### **3. Amaç:**

- Bu vektörler, kullanıcı sorgularıyla eşleştirme yapabilmek için FAISS (Facebook AI Similarity Search) kullanılarak bir arama dizini oluşturmak amacıyla kullanıldı. Kullanıcının sorgusu, log kayıtlarının TF-IDF vektörleriyle karşılaştırılarak en uygun kayıtların bulunmasına olanak tanıdı.

Bu yöntem, sorgu ile ilgili olan log kayıtlarını belirlemek için daha hassas ve anlamlı sonuçlar elde etmeyi sağladı, çünkü her log kaydındaki önemli kelimeler doğru şekilde ağırlıklandırıldı ve sık kullanılan ama anlamsız kelimelerden daha yüksek bir önem kazandı.

```

df['combined'] = df.apply(lambda row: f"{row['ip']}
{row['timestamp']} {row['method']} {row['url']}", axis=1)

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df['combined'])

print("TF-IDF matrix shape:", tfidf_matrix.shape)

tfidf_vectors = tfidf_matrix.toarray().astype('float32')

```

- `df['combined']`: IP adresi, zaman damgası, HTTP methodu ve URL'yi birleştirir.
- `TfidfVectorizer()`: TF-IDF vektörizasyonu için bir nesne oluşturur.
- `tfidf_matrix`: TF-IDF matrisini içerir.
- `tfidf_vectors`: TF-IDF matrisini numpy dizisine dönüştürür.

## **FAISS INDEKSLEME:**

FAISS (Facebook AI Similarity Search), geliştiricilerin birbirine benzeyen multimedya belgelerinin gömülü öğelerini hızla aramasına olanak tanıyan bir kütüphanedir. Karma tabanlı aramalar için optimize edilmiş geleneksel sorgu arama motorlarının sınırlamalarını çözer ve daha ölçeklenebilir benzerlik arama işlevleri sağlar. [2]

### **Bu Projede FAISS'in Rolü:**

- **Veri İndeksi Oluşturma:** TF-IDF ile dönüştürülen vektörler, FAISS kullanılarak bir indeks yapısında saklanır. Bu indeks, kullanıcı sorgularının log verileri içinde hızlıca arama yapmasını sağlar.
- **Sorgu Eşleştirme:** Kullanıcı bir sorgu girdiğinde, FAISS bu sorgunun vektörünü mevcut log vektörleriyle karşılaştırarak en yakın eşleşmeleri bulur. Bu eşleşmeler, daha sonra yanıt üretiminde kullanılır.

Sonuç olarak, FAISS, yüksek boyutlu vektörler arasındaki benzerliklerin hızlı ve verimli bir şekilde bulunmasını sağlar. Bu da kullanıcının verdiği sorgulara en uygun log kayıtlarını hızlıca getirerek, sistemin performansını önemli ölçüde artırır.

```

index = faiss.IndexFlatL2(tfidf_vectors.shape[1])
index.add(tfidf_vectors)

print("Toplam vektör sayısı:", index.ntotal)

```

- `faiss.IndexFlatL2()`: L2 mesafe metriği kullanan FAISS indeksi oluşturur.
- `index.add()`: Vektörleri FAISS indeksine ekler.

## Log Kayıtlarını Bulma:

```
def find_relevant_logs(user_query):
    date_match = re.search(r'(\d{1,2}) (\w+) (\d{4})', user_query)
    if date_match:
        day, month, year = date_match.groups()
        from datetime import datetime
        month_number = datetime.strptime(month, "%B").strftime("%b")
        search_date = f"{day}/{month_number}/{year}"
        filtered_logs = df[df['timestamp'].str.contains(search_date)]
    else:
        keywords = user_query.split()
        filtered_logs = df[df['url'].str.contains(' '.join(keywords),
case=False, na=False)]

    if filtered_logs.empty:
        return None

    log_data = []
    for _, log_entry in filtered_logs.iterrows():
        log_data.append({
            'timestamp': log_entry['timestamp'],
            'method': log_entry['method'],
            'url': log_entry['url'],
            'ip': log_entry['ip']
        })
    return log_data
```

- `find_relevant_logs(user_query)`: Kullanıcının sorgusuna göre log kayıtlarını filtreler.
- Tarih içeren sorgular için logları tarihe göre filtreler.
- Tarih içermeyen sorgular için URL'lerde anahtar kelimeleri arar.

## Yanıt Üretme:

Bu bölümde dil modeli olarak GPT-2 kullanmayı tercih ettim. Generative Pre-trained Transformer 2, OpenAI'nin büyük bir dil modelidir ve GPT modellerinin temel serisindeki ikinci modeldir. GPT-2, 8 milyon web sayfasından oluşan bir veri kümesi üzerinde önceden eğitilmiştir. [3]

### **Bu Projede GPT-2'nin Kullanımı:**

Bu projede, GPT-2 modelinin ana kullanım amacı, kullanıcıdan gelen bir sorguya dayanarak uygun bir yanıt üretmektir. İşte bu kullanımın detayları:

#### **1. Kullanıcı Sorgusu ve Log Kayıtlarıyla Yanıt Üretimi:**

Kullanıcıdan bir sorgu alındığında, ilgili log kayıtları bulunur. Bu log kayıtları, kullanıcı sorgusuyla birlikte GPT-2 modeline sunulur.

GPT-2 modeli, verilen bu metni analiz eder ve sorguya uygun bir yanıt üretir. Bu yanıt, hem kullanıcı sorgusunu hem de ilgili log kayıtlarını dikkate alarak oluşturulur.

```
model_name = "gpt2"
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

tokenizer.pad_token = tokenizer.eos_token

def generate_answer(log_data, user_query):
    log_text = "\n".join([f"{log['ip']} - {log['timestamp']} - {log['method']} {log['url']}" for log in log_data])
    input_text = f"Kullanıcı sorusu: {user_query}\n\nRelevant log records:\n{log_text}\n\n"
    inputs = tokenizer(input_text, return_tensors="pt",
max_length=512, truncation=True)
    outputs = model.generate(
        inputs['input_ids'],
        max_new_tokens=10,
        num_return_sequences=1,
        no_repeat_ngram_size=2,
        pad_token_id=tokenizer.pad_token_id,
        temperature=0.5,
        do_sample=False
    )
    answer = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return answer
```

- **GPT2LMHeadModel ve GPT2Tokenizer:** GPT-2 modelini ve tokenizer'ını yükler.
- **generate\_answer(log\_data, user\_query):** Log kayıtlarını ve kullanıcının sorusunu kullanarak modelden yanıt üretir.

### **Kullanıcıdan Soru Alma ve Yanıt Oluşturma:**

```
def answer_user_query():
    user_query = input("Please enter your query: ")
    log_data = find_relevant_logs(user_query)

    if not log_data:
        print("No relevant log entries found. Please try a different query.")
        return

    answer = generate_answer(log_data, user_query)
    print("Generated Answer:")
    print(answer)

answer_user_query()
```

**answer\_user\_query():** Kullanıcıdan bir sorgu alır, uygun log kayıtlarını bulur ve bu loglardan yanıt üretir.

## Performansın Değerlendirilmesi:

Ödevde verilen talimatlara göre ilerleyip hazırladığım bu proje, belli başlı zamanla alakalı sorulara doğru yanıt verirken bazı spesifik sorularda yetersiz kalabiliyor. Özellikle loglarla alakalı sorularda bir nebze daha iyi yanıt oluşturabiliyor. Aynı zamanda sorularımı İngilizce sorduğumda daha alakalı yanıtlar alabildiğimi fark ettim.

```
Please enter your query (or type 'exit' to quit): is there any access to dashboard?
Generated Answer:
Kullanıcı sorusu: is there any access to dashboard?

Relevant log records:
10.106.5.39 - 12/Aug/2024:23:26:45 +0300 - GET /dashboard/
10.106.5.39 - 12/Aug/2024:23:26:46 +0300 - GET /dashboard/
10.200.17.116 - 13/Aug/2024:09:15:03 +0300 - GET /dashboard/
10.200.17.116 - 13/Aug/2024:09:15:03 +0300 - GET /dashboard/
10.106.5.39 - 16/Aug/2024:03:14:19 +0300 - GET /dashboard/
10.106.5.39 - 16/Aug/2024:03:14:19 +0300 - GET /dashboard/
10.200.17.105 - 16/Aug/2024:09:27:00 +0300 - GET /dashboard/
10.200.17.105 - 16/Aug/2024:09:27:00 +0300 - GET /dashboard/

11.08.2016 - 11/08
Please enter your query (or type 'exit' to quit): |

Generated Answer:
Kullanıcı sorusu: what pages were accessed on 13 August 2024?

Relevant log records:
10.200.17.116 - 13/Aug/2024:09:15:03 +0300 - GET /
10.200.17.116 - 13/Aug/2024:09:15:03 +0300 - GET /dashboard/
10.200.17.116 - 13/Aug/2024:09:15:03 +0300 - GET /
10.200.17.116 - 13/Aug/2024:09:15:03 +0300 - GET /dashboard/
10.200.17.116 - 13/Aug/2024:09:15:10 +0300 - GET /HNAP1/
10.200.17.116 - 13/Aug/2024:09:15:10 +0300 - GET /
10.200.17.116 - 13/Aug/2024:09:15:10 +0300 - GET /
10.200.17.116 - 13/Aug/2024:09:15:17 +0300 - GET /DeviceDescription.xml
```

## Modelimin yanıt verebildiği sorulardan birkaçı:

Is there any Access to dashboard?

What pages were accessed on 13 August 2024?

Which IP addresses accessed the server the most on a specific date?

Which IP address made the most GET requests?

Bazı soruları daha modele sorarak ve deneyerek sistemin log kayıtlarına dayalı olarak sorulara yanıt verme yeteneğini test ettim. Ancak bazı sorunlar var gibi, özellikle modelin doğru ve alakalı yanıtlar veremediği durumlar:

1. Yanıtlarda Eski Tarihler: Örneğin, `17/Jul/1916` gibi alakasız tarihlerle karşılaşıldı. Bu durum, modelin log kayıtlarını düzgün işleyemediğini veya yanıt oluştururken tutarsızlıklar olduğunu gösterebilir.

2. Yanıtlarda Eksik veya Yanlış Bilgiler: Sorduğum sorulara karşılık gelen log kayıtları her zaman tam olarak listelenmemiş ya da doğru bilgiler verilmemiş. Örneğin, `What are the most frequently accessed URLs?` sorusuna karşılık hiçbir kayıt bulunamadığını belirtti.

3. Yanıtın Tamamlanmaması: Bazı durumlarda yanıtın ortasında veya sonrasında modelin oluşturduğu sonuç kesilmiş gibi görünüyor, örneğin "The following logs are from the following IP addresses" ifadesiyle başlayan ancak devam etmeyen yanıtlar oluştu.

Bu Sorunları Çözmek İçin Yapılabilecekler:

1. Log Kayıtlarının Daha İyi Analizi: Log kayıtlarını işlerken tarih, IP adresi, URL gibi alanların düzgün şekilde ayrıştırılabilir. Ayrıca, sorularımıza yanıt verirken doğru ve tutarlı formatta log kayıtları döndüğünden emin olmak için modelin çıktısını gözden geçirebiliriz.

2. Model Parametrelerinin Ayarlanması: Modelin yanıtlarını iyileştirmek için örneğin `do\_sample=True` ve `temperature` ayarlarını optimize edebiliriz. Bu, yanıtların daha çeşitli ve uygun olmasına yardımcı olabilir.

3. Soru-Model İletişimini İyileştirme: Modelin soruları nasıl algıladığını ve yanıt ürettiğini dikkatlice incelememiz gerekebilir. Gerekirse soruların formatını ve modelin bu sorulara verdiği yanıtları daha spesifik hale getirmek için ileriki zamanlarda kodu güncelleyebiliriz.

4. Ek Log Verisi ve Sorularla Test Etme: Farklı tarihler ve IP adresleri içeren log kayıtlarıyla sistemimi daha fazla test etmeliyim. Böylece sistemin farklı durumlarda nasıl yanıt verdiğini gözlemleyebilir ve gerekli iyileştirmeleri yapabilirim.

## KAYNAKLAR

1. <https://dergipark.org.tr/en/download/article-file/1216978#:~:text=TF%2DIDF%2C%20bir%20dok%C3%BCmandaki%20herhangi,aras%C4%B1daki%20anlamsal%20ili%C5%9Fkiyi%20ifade%20edememektedir.>
2. <https://ai.meta.com/tools/faiss/>
3. <https://en.wikipedia.org/wiki/GPT-2>