

1. Giriş

Bu proje, başlangıçta klasik veri analizi ve işleme yöntemlerine dayanarak yapılandırılmış veriler üzerinde anlamlı sonuçlar elde etmeyi amaçlayan bir proje olarak başladı. İlk adımda, sanal bir veri seti oluşturuldu ve bu veri üzerinde çeşitli veri temizleme ve ayıklama işlemleri gerçekleştirildi. Ardından, bu yapılandırılmış verileri embedding modelleri kullanılarak tüm CSV dosyası vektörleştirildi ve FAISS üzerinde depolandı. Bu süreçte, kullanıcının doğal dilde sorularına anlamlı cevaplar üretebilmek için OpenAI'nin GPT-3.5-turbo modeli entegre edildi.

Ancak, yapılandırılmış verilerden elde edilen sonuçlar, yapılandırılmamış verilerdeki gibi tatmin edici olmadı. Bu yöntem, beklenenin aksine, CSV gibi yapılandırılmış veriler üzerinde etkili sonuçlar vermedi. Yapılandırılmış verilerin zaten bir anlam taşıyan kategorik yapısı, vektörleştirildiğinde kayboluyor ve sonuçların doğruluğu önemli ölçüde düşüyordu. Bu hayal kırıklığı, projenin yönünü değiştirdi ve yeni bir çözüm arayışına yöneltti.

Araştırmalarım, yapılandırılmış verilerde yüksek başarı elde etmenin anahtarının, sorgu tabanlı bir yaklaşıma dayanması gerektiğini gösterdi. LLM (Large Language Model) teknolojilerinin sorgu üzerinden daha etkili çözümler sunabileceğini fark ettim. Bu yeni yaklaşımda, vektörleştirme işlemi yalnızca kullanıcının sorgusuna göre en alakalı sütunları belirlemek için kullanıldı. Örneğin, veri setinde "Konum", "HTTP Yöntemi" ve "Durum Kodu" gibi sütunlar bulunuyorsa ve kullanıcı "Kaç farklı mesken vardır?" şeklinde bir soru soruyorsa, RAG (Retrieval-Augmented Generation) teknolojisi konum ve mesken arasında anlamlı bir bağ kurdu. Sistem, sorguyu konum sütununa yönlendirdi ve OpenAI GPT-3.5-turbo modelinin desteğiyle doğru yanıtı üretebildi.

Bu yeni yaklaşım, yapılandırılmış verilerle çalışırken performans ve doğruluk açısından çok daha başarılı sonuçlar sağladı. Bu proje, geleneksel veri analizi ve vektörleştirme yöntemlerinin ötesine geçerek, LLM tabanlı sorgu teknolojisinin nasıl entegre edilebileceğini göstermektedir.

Projenin Gerçek Zamanlı Test Linki: <https://cs8q9jzbzelbehrgbvuuu.streamlit.app> (eğer sisteme girdiğinizde çökmüş olursa lütfen bana ulaşın çünkü streamlitin 1GB Free hakkı var ve model çalışıp çıktı ürettikçe hafıza doluyor ve “out of memory” gibi hatalar ile projeyi çöktürüyor.)

Projenin Test Video Linki(Saf Sorgu): <https://youtu.be/EWYw0yaa3EA>

Projenin Test Video Linki(Veri Görselleştirme): <https://youtu.be/If0q3UztiHY>

2. Proje Adımları

2.1. Gereksinimlerin Belirlenmesi ve Ortamın Hazırlanması

Projenin başlangıcında, semantik arama işlevselliği sağlamak için uygun teknolojilerin belirlenmesi kritik bir adımdı. Bu süreçte FAISS, SentenceTransformers, OpenAI'nin GPT modelleri ve Llama Index gibi teknolojilerin uygun olduğuna karar verildi. Ayrıca, kullanıcı arayüzü için Streamlit platformunun kullanılması kararlaştırıldı.

2.2. CSV Dosyasının Y  klenmesi ve İlk İşlemeler

İlk olarak, kullanıcının y  klediđi CSV dosyasının uygun bir řekilde işlenmesi sađlandı. Y  klenen veriler bir pandas DataFrame olarak okunup, kullanıcının g  r  nt  lemesine olanak tanındı. Bu ařama, veri   zerinde yapılacak işlemler i  in temel teřkil etti.

2.3. Semantik Arama İ  in FAISS ve SentenceTransformers Kullanımı

Uygulamanın ana işlevselliđi, CSV dosyasındaki s  t  n adlarının semantik olarak vekt  rleřtirilmesi ve FAISS kullanılarak bu vekt  rlerin saklanması   zerine kuruldu. Kullanıcının sorgusu da aynı y  ntemle vekt  rleřtirilerek, FAISS ile en alakalı s  t  nlar belirlendi. Bu adım, kullanıcının dođal dilde yazdıđı sorgularla, verideki en uygun s  t  nlar   zerinden arama yapabilmesini sađladı.

2.3.1. En İyi Embedding Modelinin Se  imi

Bu projede kullanılan embedding modeli, sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 modelidir. Bu model, T  rk  e de dahil olmak   zere bir  ok dilde etkili performans g  stermesiyle bilinir. Diđer bir  ok embedding modeli,   zellikle T  rk  e gibi dil bilgisel yapısı karmařık dillerde zayıf sonu  lar verebilir.   rneđin, bert-base-multilingual-cased gibi diđer modeller, T  rk  e'deki ince ayrımları ve anlam iliřkilerini yeterince yakalayamadıđından, semantik arama ve RAG (Retrieval-Augmented Generation) gibi uygulamalarda performans sorunları yařayabilir. Bu nedenle, T  rk  e dilinde y  ksek dođruluk sađlayan ve d  ř  k gecikme sunan paraphrase-multilingual-MiniLM-L12-v2 modelinin se  ilmesi, projenin bařarısı a  ısından kritik bir rol oynadı.

2.4. LLM Entegrasyonu ile Pandas Sorgusu

En alakalı s  t  nlar belirlendikten sonra, LLM desteđi ile bu s  t  nlar   zerinden bir Pandas sorgusu oluřturuldu. Bu sorgu, kullanıcının isteđine uygun olarak dinamik bir řekilde oluřturuldu ve sonu  lar kullanıcıya sunuldu. Bu adımda OpenAI'nin GPT-3.5-turbo modeli kullanıldı.

2.5. QueryPipeline ile Sorgu Akışının Y  netimi

Sorgu akışının daha sistematik bir řekilde işlenmesi amacıyla Llama Index'in QueryPipeline yapısı kullanıldı. Bu yapı, kullanıcının sorgusunu işleyerek uygun bir řekilde sonu     reten adımları y  netti ve sonucun kullanıcıya sunulmasını sađladı.

3. Karřılařılan Zorluklar

3.1. Vekt  r Veritabanları ve Yapılandırılmıř Veriler

Projenin bařında, CSV dosyasındaki verilerin tamamını vekt  rleřtirerek bunları FAISS ile saklayıp, kullanıcı sorgusuna uygun sonu  ları getirme y  ntemi denendi. Ancak bu yaklařımın bazı ciddi kısıtlamaları olduđu fark edildi:

3.1.1. Vekt  r Veritabanları Neden Yapılandırılmıř Veriler İ  in Harika Deđil?

Vektör veritabanları, yapılandırılmamış verilerde harika sonuçlar üretirler. Örneğin, metin, görsel veya ses gibi veri türlerinde, bu verileri vektör uzayına taşıyarak semantik yakınlıklarını hesaplamak mümkündür. Ancak yapılandırılmış veriler (örneğin, bir CSV dosyasındaki sütunlar) zaten anlamlı bir yapı taşır ve bu yapıyı korumak önemlidir.

Yapılandırılmış verilerde, her sütun genellikle belirli bir özellik veya kategoriye karşılık gelir ve bu sütunların birbirleriyle olan ilişkileri belirgindir. Bu nedenle, bu verileri vektörleştirip FAISS gibi bir vektör veritabanına koymak, bu anlamlı yapıyı kaybetmeye yol açabilir. Ayrıca, bu veritabanları genellikle belirli bir sorguya göre en yakın vektörleri getirmek üzerine tasarlanmıştır; ancak bir CSV dosyasındaki tüm sütunlar semantik olarak anlamlı olmayabilir. Bu da veritabanından alınan sonuçların tutarlılığını ve doğruluğunu etkileyebilir.

3.1.2. Performans ve Sonuçlar

RAG teknolojisi, bu kısıtlamaların üstesinden gelmek için kullanıldı. Kullanıcı sorgusuna göre en alakalı sütunları belirleyip, yalnızca bu sütunlar üzerinden bir sorgu yapılması daha etkili bir yaklaşım olarak belirlendi. Örneğin, bir CSV dosyasında "Konum", "HTTP Yöntemi" ve "Durum Kodu" gibi sütunlar bulunduğunu düşünelim. Eğer kullanıcı "Kaç farklı mesken vardır?" diye bir soru sorarsa, RAG teknolojisi, "mesken" ve "konum" arasında anlamlı bir ilişki kurar. Bu ilişki kurulduktan sonra, sistem Konum sütunundan sorgu yaparak, OpenAI GPT-3.5-turbo modelinin yardımıyla bu sorguya uygun bir yanıt üretir. Bu yaklaşım hem daha doğru sonuçlar sunar hem de performans açısından büyük avantaj sağlar.

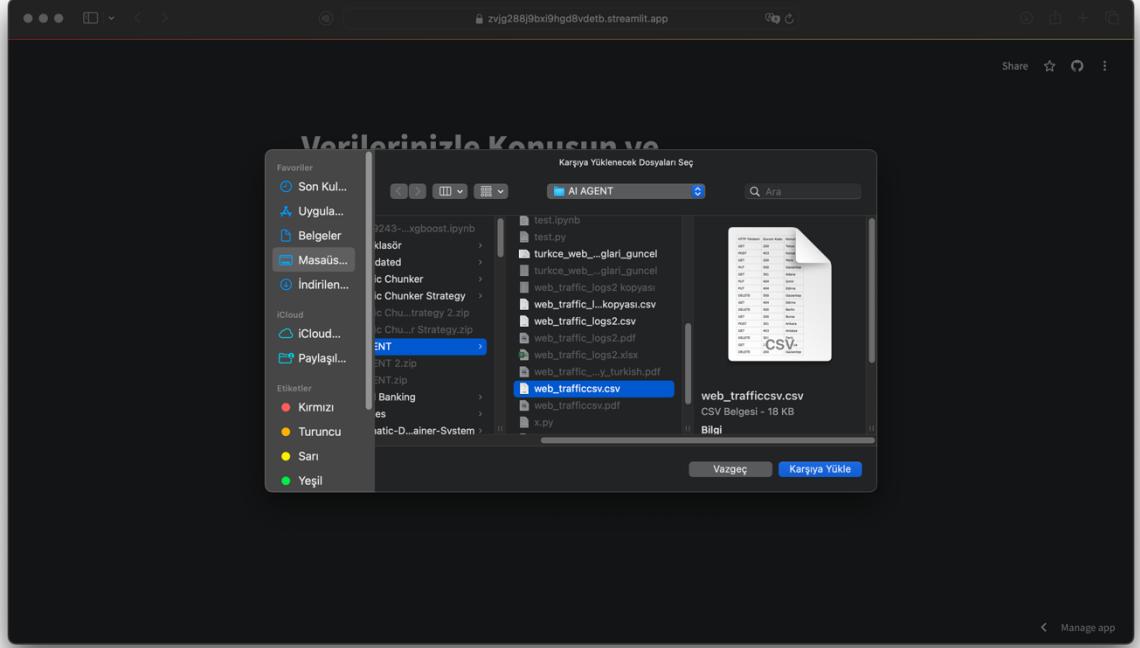
4. Oluşturulan Sistemin Kullanım Kılavuzu

4.1 Veri Yükleme

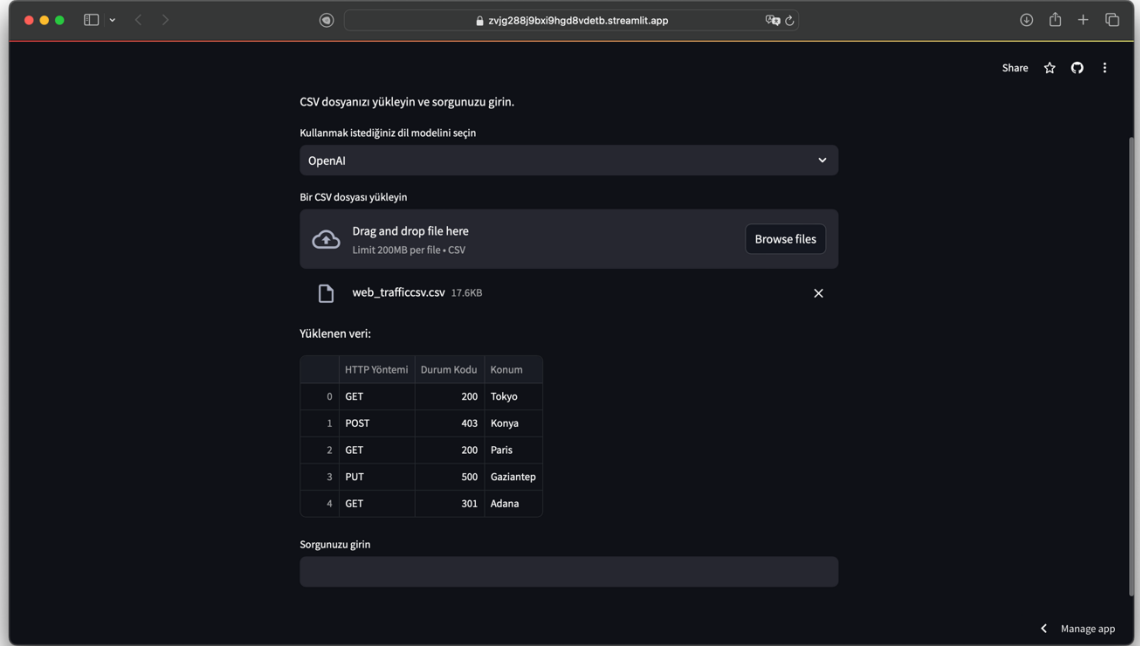


Şekil 1.

Şekil 1.'de görülen sistemin arayüzüdür. Bu arayüzde model seçimi yapılan kısımdan istediğiniz modeli seçmenize olanak tanınmaktadır. Şu anda kısıtlı bellek açısından sadece OpenAI-3.5-Turbo modelinin API entegrasyonu bulunmaktadır. İlgili sunucu tedariki sonucu açık kaynaklı LLM modelleri de kolayca entegre edilebilmesi için yuvası yapılmıştır. Model seçimi yapıldıktan sonra “Browse Files” kısmından tarafınıza gönderilen csv formatındaki dosyayı yüklemelisiniz. Yüklenen dosya Şekil 2.'de gösterilmiştir.



Şekil 2.

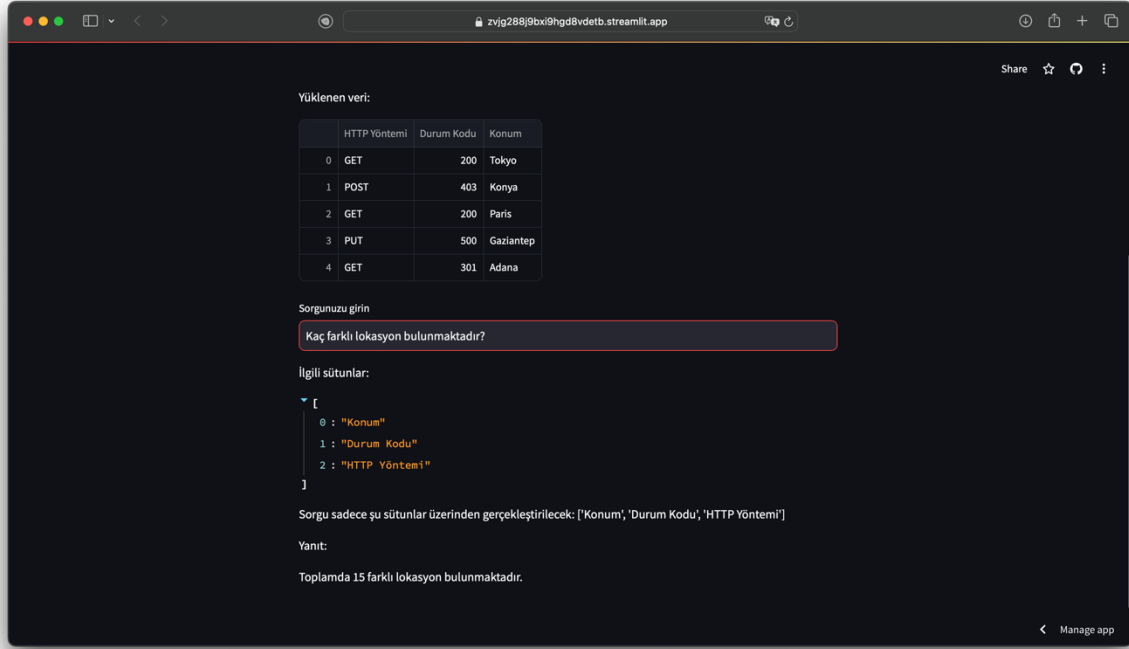


Şekil 3.

Veriyi başarıyla yükledikten sonra Şehil 3.'de de görüldüğü üzere kullanıcıdan sorgu alabilmek için bir sorgu balonu açılacaktır. Bu süreçten sonra açılan sorgu balonuna istediğiniz sorguyu girerek sonuçlarını inceleyebilirsiniz.

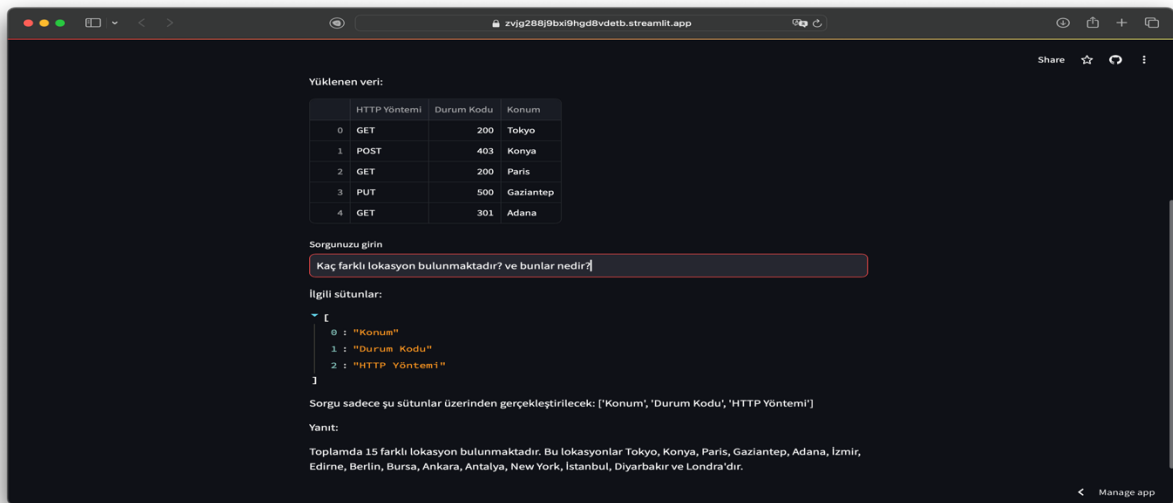
5. Performans ve Doğruluk Değerlendirmesi

5.1. Çıktı Kalitesinin ve Doğruluğun Değerlendirmesi



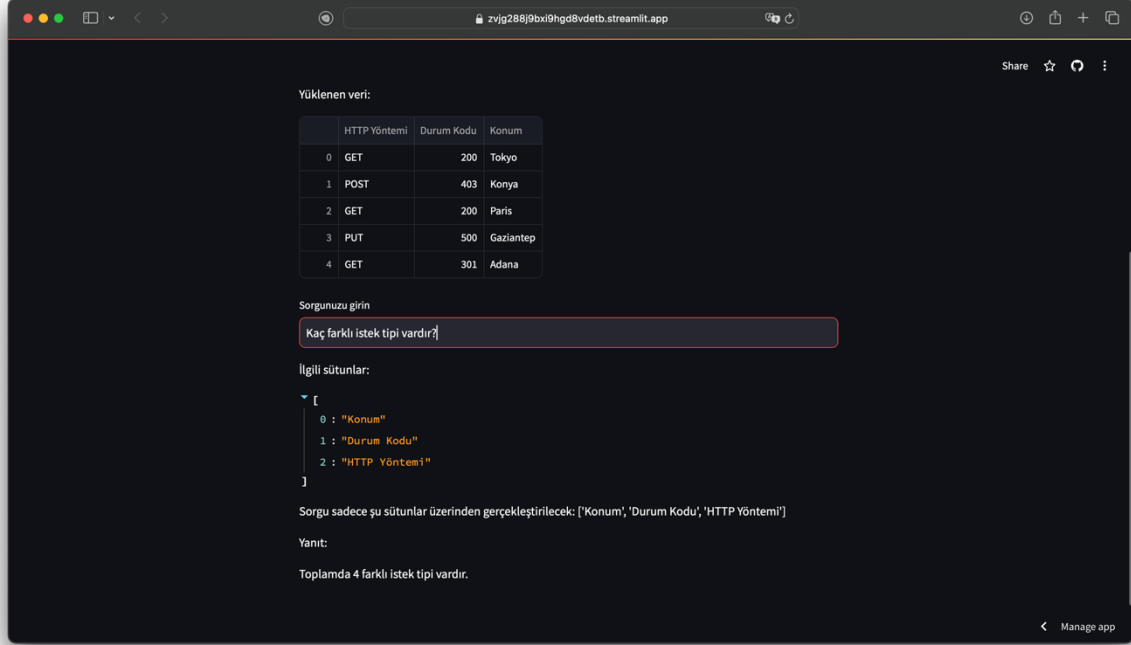
Şekil 4.

Şekil 4.'de görüldüğü üzere “Kaç farklı lokasyon bulunmaktadır?” sorgusuna modelin yanıtı “Toplamda 15 farklı lokasyon bulunmaktadır” olmuştur. Bu model yanıtı hem zengin hem de %100 doğruluğa sahiptir.



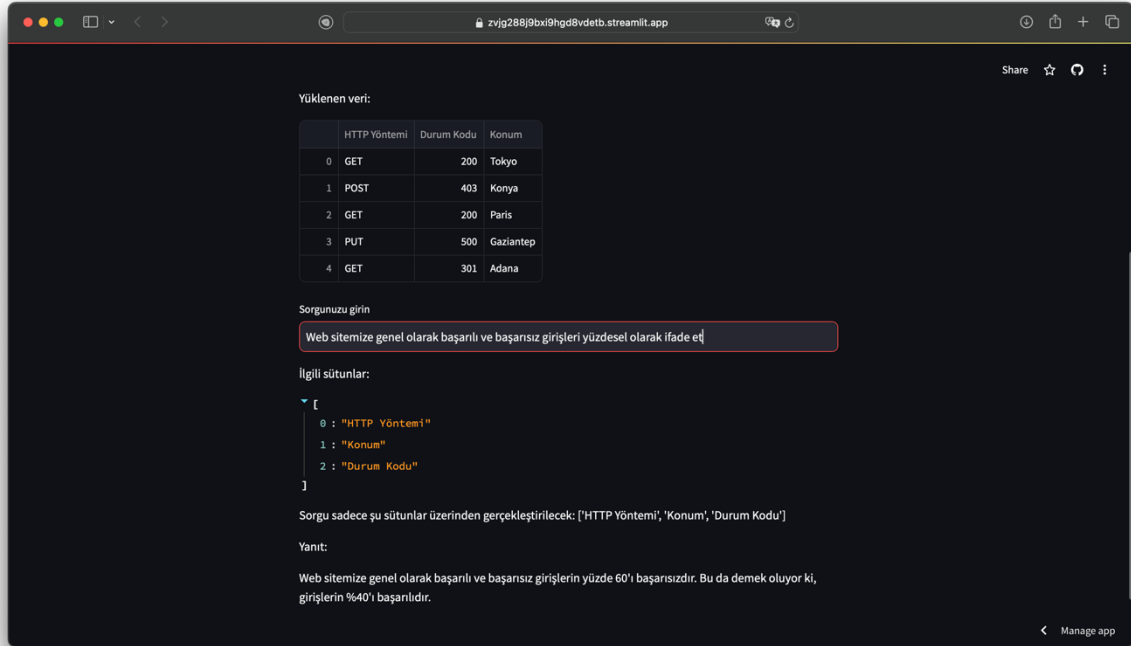
Şekil 5.

Şekil 5.'te bir önceki sorguyu daha detaylandırarak konum isimlerini sormuşuz ve model yine doğru bir cevap vermiştir.



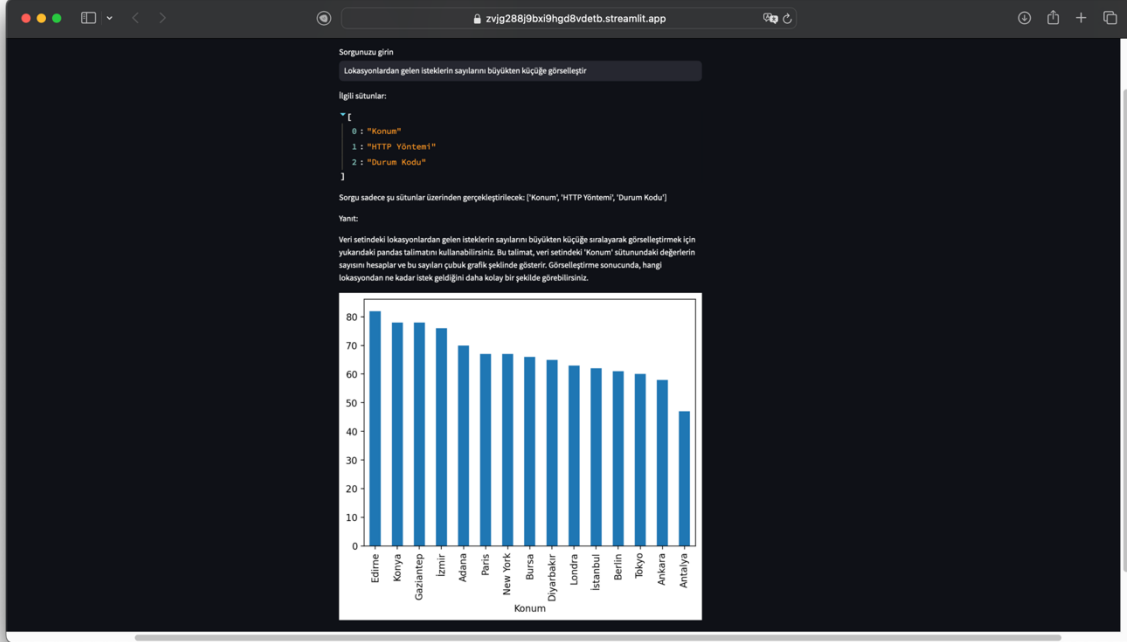
Şekil 6.

Şekil 6. İncelendiğinde bu sefer veri setimizdeki HTTP Yöntemlerini kastederek “Kaç farklı istek tipi vardır?” diye bir soru sorduk ve sistem FAISS vektör veri tabanında kullanıcıdan gelen sorguya göre en yakın sütunları bulup sorguyu ilgili sütunlarda gerçekleştirip başarılı bir sonuç çıktısı üretti.



Şekil 7.

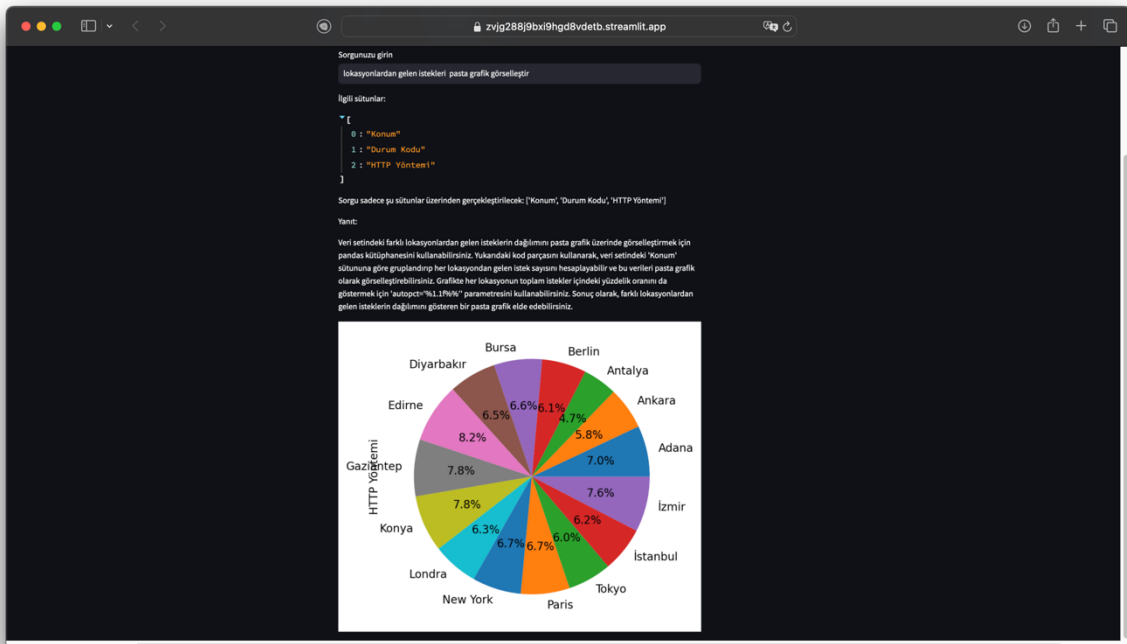
Şekil 7. incelendiğinde sorguyu biraz daha çeşitlendirerek web sitemizin loglarından oluşan veri setimize “Web sitemize genel olarak başarılı ve başarısız girişleri yüzdesel olarak ifade et” diye bir sorgu gönderdik ve genel olarak web sitemize girişlerin problemlili mi yoksa problemsiz mi olduğunu sorgulamak istediğimizi düşünelim. Modelin çıktısı yine son derece doğru.



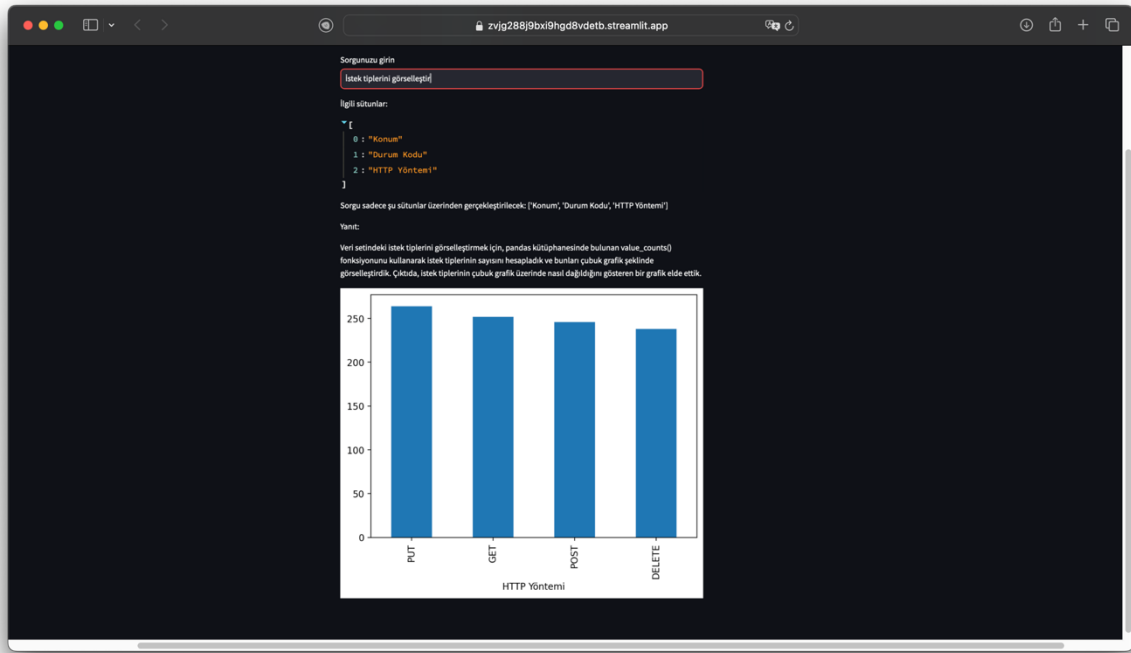
Şekil 8.

Normal şartlarda tarafımıza gönderilen görevde kullanıcıdan girdiye göre verileri görselleştirme gibi bir görev yoktu fakat bu projeyi yaparken projenin çok daha farklı boyutlara gelebileceğini fark edip geliştirmelere devam ettim. Şekil 8. incelendiğinde “Lokasyonlardan gelen isteklerin sayılarını büyükten küçüğe görselleştir” sorgusu girilmiş ve model başarıyla bir görselleştirme yaparak çıktı grafiğini üretmiştir. Bu sayede yüklenen veri setinden istediğimiz görselleştirmeyi de yapabilir bir standarda erişmiş bulunduk.

Şekil 9.

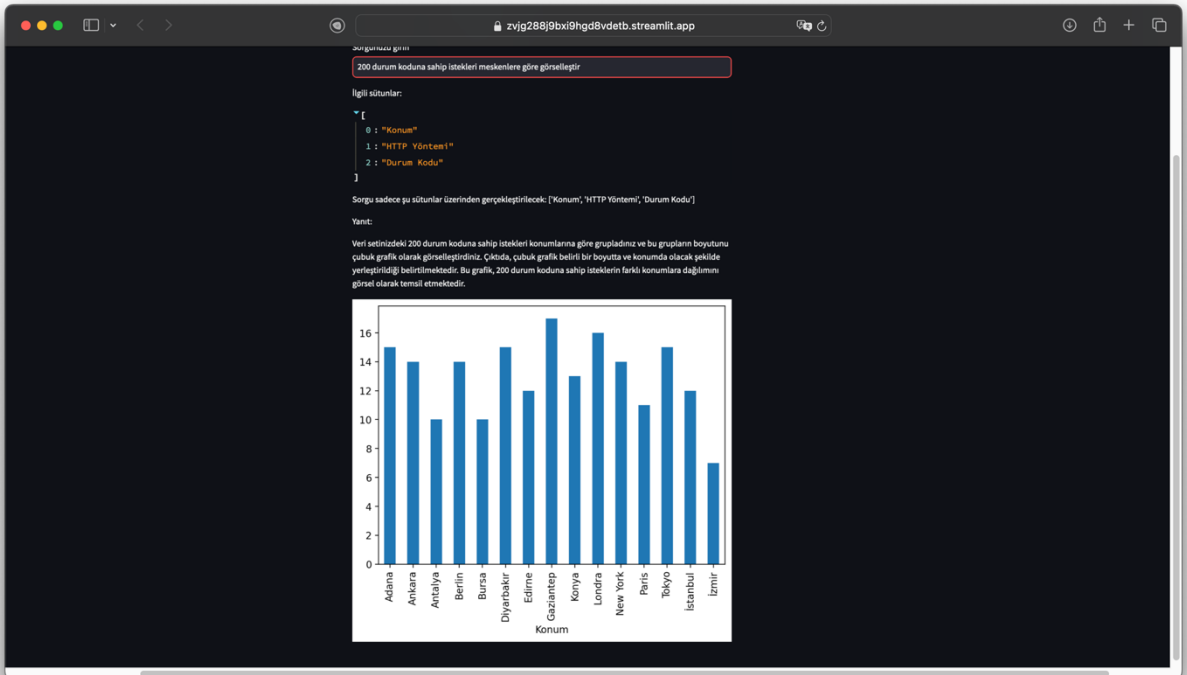


Şekil 9. incelendiğinde farklı görselleştirme teknikleri ile de görselleştirebildiğimizi ve aynı zamanda model tarafından ilgili grafiğe bir açıklama da getirdiğini görebiliyoruz.



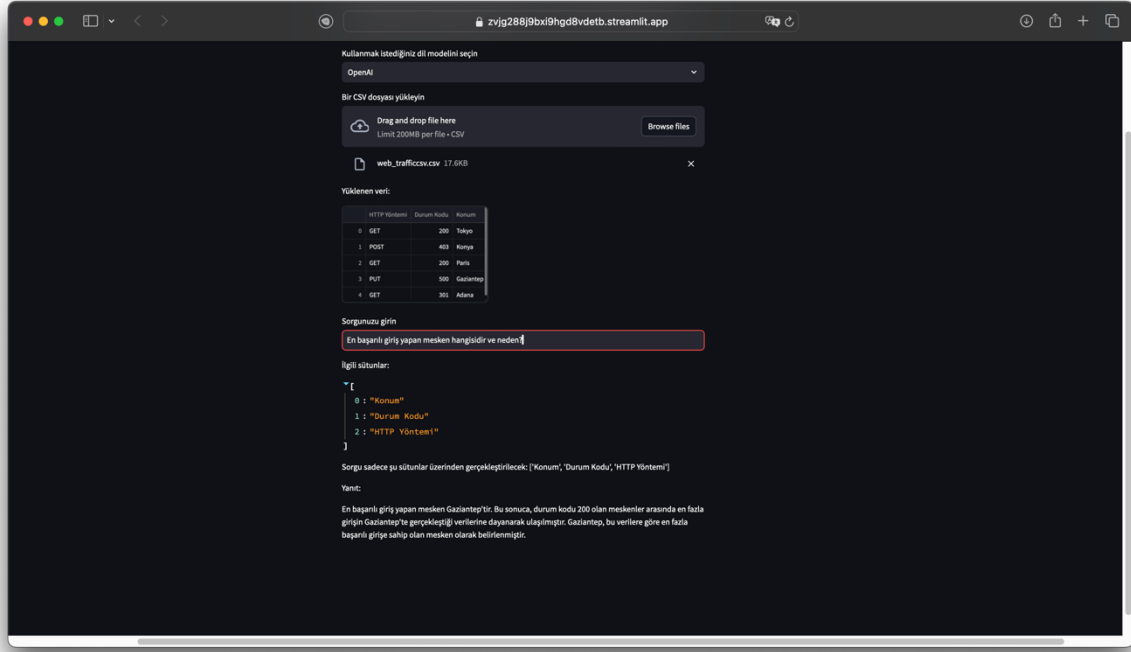
Şekil 10.

Şekil 10. incelendiğinde farklı sütunlar üzerinde de görselleştirme yapabileceğimizi görebiliyoruz.



Şekil 11.

Şekil 11. incelendiğinde karmaşık sorgularda da yüksek performans ve kaliteli bir görselleştirme ürettiğini gözlemleyebiliyoruz.



Şekil 12.

Şekil 12. incelendiğinde veri setine genel bir soru sorulmuş ve nedeni açıklattırılmış. Modelin çıktısı zengin, çeşitli ve son derece anlamlı.

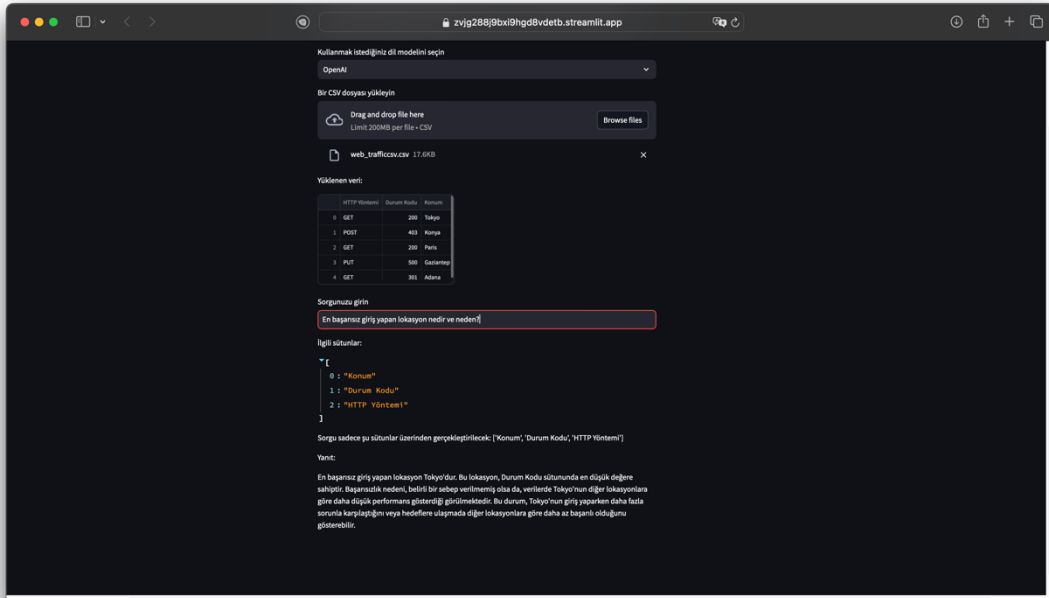
Yapılan model tarafından yanıtların saniye bazında analizi incelendiğinde;

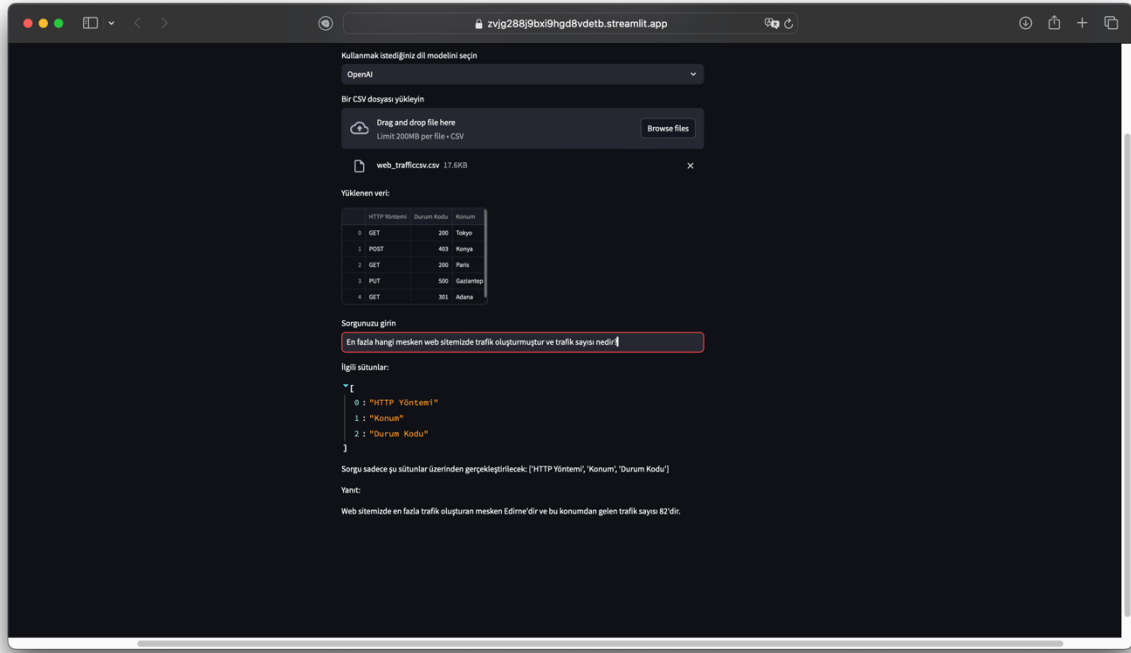
Saf Sorgular: 1-1,73 saniye

Grafik Sorgular: 1,25-2 saniye

Modelin çıktı hızları analiz edildiğinde hızlı, zengin, kaliteli ve doğru yanıtlar verdiği gözle görülmektedir.

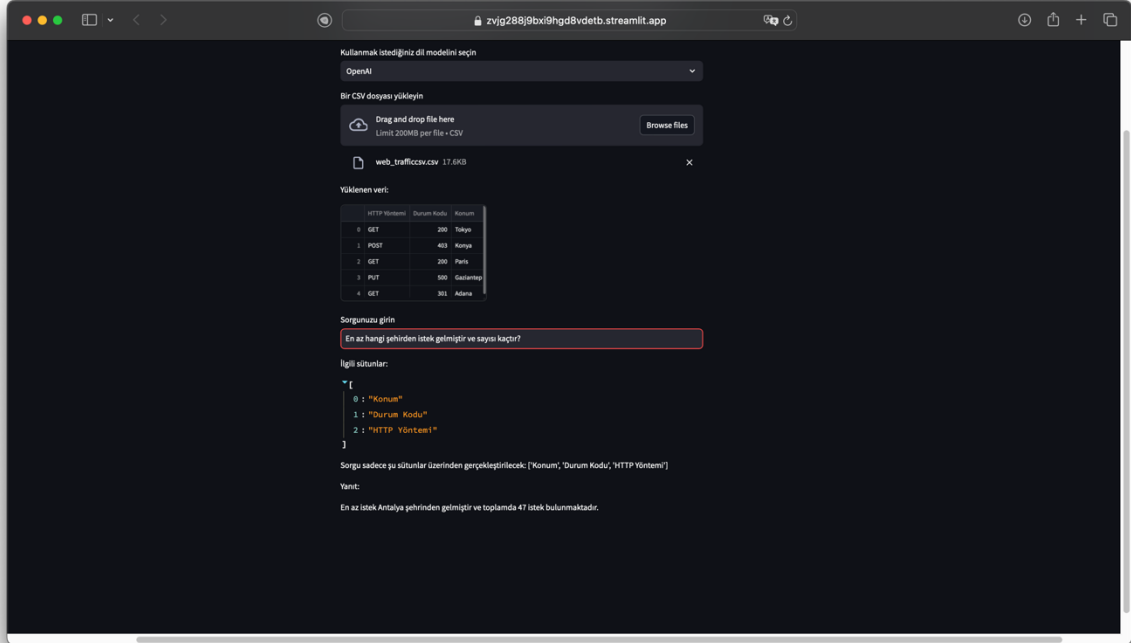
5. Diğer Sorgular





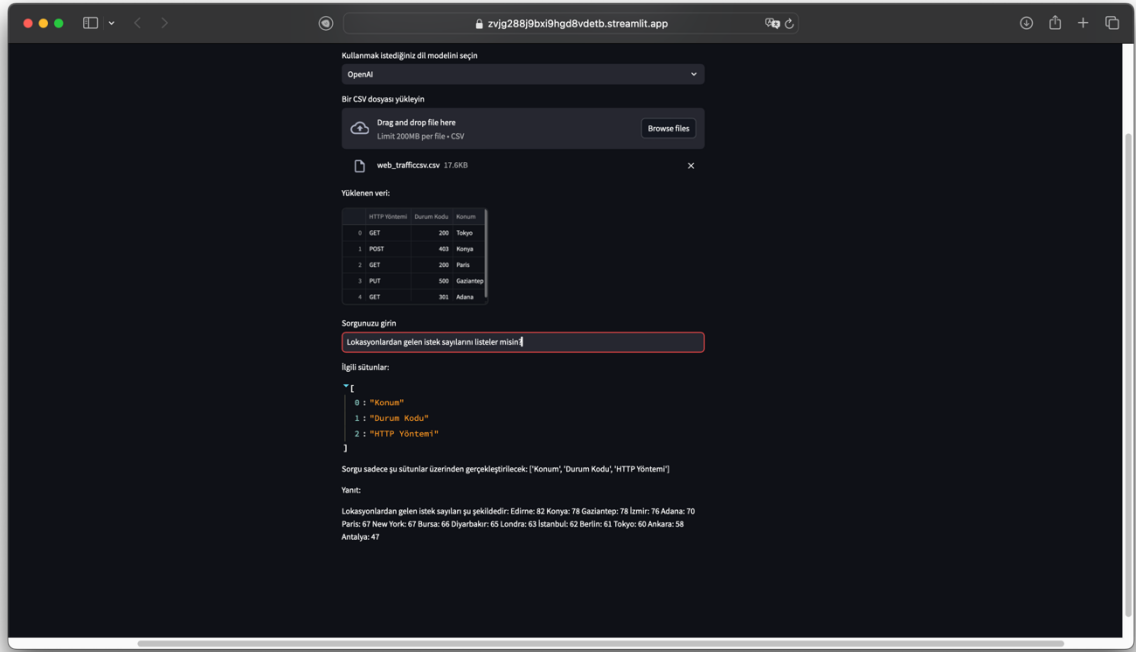
Şekil 14.

Şekil 14. İncelendiğinde trafik sayısı gibi özel sorgulara kapsamlı bir anlamlandırma yaparak doğru ve kaliteli cevabı model üretmiş bulunmaktadır.



Şekil 15.

Şekil 15. incelendiğinde Konumlar bazında gelen istek sayılarının “en az” olanını sorgulayıp doğru ve kaliteli bir cevap almış bulunmaktayız.



Şekil 16.

Son olarak “Lokasyonlardan gelen istek sayılarını listeler misin?” sorgusu ile lokasyon bazında gelen toplam istekleri gruplayıp modelimiz kaliteli ve doğru şekilde bir çıktı üretmiştir.

6. Sonuç ve Gelecek Çalışmalar

Bu proje, yapılandırılmış veriler üzerinde semantik arama yapmanın zorluklarını ve bu zorlukların nasıl aşılabileceğini göstermiştir. Vektör veritabanlarının yapılandırılmış verilerde neden verimli olmadığını anlamak, bu projede izlenen çözüm yollarını şekillendirmiştir. Kullanıcı girdisine göre sütunların seçilmesi ve yalnızca bu sütunlar üzerinde sorgulama yapılması, sistemin performansını ve doğruluğunu artırmıştır.

Gelecek çalışmalarda, daha gelişmiş modelleme teknikleri kullanarak, yapılandırılmış veri setleri üzerinde daha karmaşık sorgulamalar yapılması hedeflenebilir. Ayrıca, farklı türdeki veri setleri üzerinde bu yaklaşımın uygulanabilirliği araştırılabilir. Embedding sonucu vektör veri tabanında yapılan sorgunun kalitesini arttırabilmek adına Türkçe kelimeler ile bir açık kaynaklı embedding modeli fine-tune edilerek Türkçeye özel kaliteli bir embedding modeli üretilebilir. Kullanıcıdan alınan geri dönüşler ile cevabın kalitesi ölçülüp modelin performans güncellemeleri için kullanılabilir.