Contents

Chapter 1

Teknik Görev Dokümantasyonu

1.1 Görev 4: RAG Tabanlı Teknik Servis Bilgi Asistanı – rag-supportbot

1.2 Projenin Amacı

Bu proje, bir teknik servis firmasının çağrı merkezi ve destek personelinin sık karşılaştığı müşteri sorularını, önceden hazırlanmış belgelerden çekerek cevaplayan bir yapay zeka destekli bilgi asistanı geliştirmeyi amaçlar.

Sistemin temel felsefesi, klasik LLM'lerin (Large Language Models) ezbere cevaplar üretmesindense, kaynağa dayalı ve güvenilir cevaplar üretmesini sağlamak. Bunun için Retrieval-Augmented Generation (RAG) mimarisi tercih edilmiştir.

RAG mimarisi sayesinde:

- Her cevap, gerçekten var olan bir belge parçasına dayanır.
- Kullanıcıya hem yanıt hem de o yanıtın geldiği belge referansı gösterilir.
- AI'ın uydurma (hallucination) riski azaltılır.
- Destek personelinin yükü hafifler, bilgiye erişim süresi kısalır.

1.3 Önceki Görevlerle Bağlantı

Görev	Açıklama	Bu Projeyle İlişkisi
Task 1	Teknik servis web sitesi (Next.js)	Bot, bu siteye entegre edilerek interaktif destek sağlar
Task 2	İptal taleplerine otomatik karar sistemi	Belge içinde iptal koşulları geçtiği için destekleyici rol oynar
Task 3	Google Ads anahtar kelime sınıflandırıcı	Kullanıcı soruları sınıflandırılarak doğru belgeye yönlendirilebilir

1.4 Kullanılan Teknolojiler ve Gerekçeleri

Katman	Teknoloji	Neden Bu Teknoloji Seçildi?
LLM	LLaMA 3 Instruct (GGUF)	Ücretsiz, açık kaynak, cihaz üstü çalışabilen güçlü bir model. Gizlilik gereksinimlerine uygun. Projede kurulu.
Embedding Model	nomic-embed-text / Instructor-xl	Normalize vektör çıktısı verir. FAISS ve LlamaIndex ile uyumludur. Ücretsiz ve offline çalışabilir.
Vektör Veritabanı	FAISS + LlamaIndex	FAISS hızlı ve hafif bir vektör veritabanıdır. Metadata destekli aramalar için LlamaIndex entegrasyonu eklenmiştir.
RAG Orkestrasyonu	LlamaIndex + LangChain	LlamaIndex, temel RAG pipeline için tercih edilir. Gelişmiş senaryolarda (query rewriting, document grading, akıllı fallback) LangChain zincir yapısı ile desteklenir.
Frontend UI	Streamlit	Hızlı prototipleme, sade görsellik, etkileşimli form desteği. Teknik olmayan kullanıcılar için uygundur.
(Opsiyonel) Backend API	FastAPI	Sistem servisleştirildiğinde frontend harici kullanımlar için REST API desteği sağlanabilir.
Veri Formatı	.txt, .pdf, .md	Gerçek dünyada teknik belgelerin bulunduğu tipik formatlar. Şu anda 5 adet `.txt` test dokümanı mevcut.

Not: Tüm işlemler llama.cpp altyapısı ile tamamen cihaz üstünde (offline) çalışır. Ücretli API veya bulut servisi gerekmez.

Ek Not – Uygulama Bağımlılıkları (requirements.txt)

Yukarıdaki teknolojilerin Python tarafında sorunsuz çalışabilmesi için aşağıdaki

modüller `requirements.txt` dosyasına eklenmiştir:

- `llama-index==0.9.48` → LlamaIndex'in kararlı ve modüler olmayan sürümü. `SimpleDirectoryReader`, `VectorStoreIndex`, `ServiceContext` gibi bileşenler bu sürümde doğrudan kullanılabilir.

- `sentence-transformers` → Embedding modelleri (`nomic-embed-text`, `instructor-xl`, `all-MiniLM`) için HuggingFace temelli çalışma sağlar.

- `faiss-cpu` → Cihaz üstü FAISS vektör arama motoru. FAISS, LlamaIndex ile birlikte kullanılmak üzere buradan çağrılır.

- `tqdm` → Embedding ve veri yükleme işlemleri sırasında kullanıcıya progress bar (ilerleme göstergesi) sağlar.

Not: Daha yeni `llama-index` sürümleri (ör. 0.13.0) modüler yapıya geçtiğinden import hataları oluşmuştur. Bu nedenle `0.9.48` sürümüne geri dönülmüştür.

1.5 Proje Dizini: `rag-supportbot/`

```
rag-supportbot/
├── data/ # Belgeler (TXT, PDF, MD)
├── app/ # Tüm işlevsel modüller
│ ├── embedder.py # Belgeleri parçalayıp embedding'e çeviren modül
│ ├── retriever.py # FAISS + LlamaIndex ile benzer belge getirme
│ ├── llm_generator.py # LLM'e bağlanarak cevap üretir
│ ├── feedback_logger.py # Kullanıcı geri bildirimlerini loglar (JSONL)
│ ├── langchain_wrappers/ # Gelişmiş LangChain zincirleri
│ │ ├── query_rewriter_chain.py # Sorguyu daha anlamlı hale getirir
│ │ ├── document_grader_chain.py # Belgenin soruya uygunluğunu değerlendirir
│ │ ├── fallback_router_chain.py # Belge bulunamazsa fallback akışını tetikler
│ │ └── rag_chain.py # End-to-end RAG akışı zinciri
│ └── ui_streamlit.py # Kullanıcı arayüzü (Streamlit tabanlı)
├── db/ # FAISS dizini (vektör dosyaları burada tutulur)
├── logs/ # Kullanıcı log dosyaları (.jsonl)
├── notebooks/ # Testler ve deneysel prototipleme (Jupyter)
├── requirements.txt # Python bağımlılık listesi
└── demo.ipynb # Proje sunumu ve demo içeriği
```

1.6 Teknik Stratejiler

1.6.1 Chunking (Parçalama)

Belgeler, yaklaşık 500 token'lık anlamlı bölümlere ayrılır. Bu yöntem, embedding kalitesini artırır ve belge içeriğinin LLM'e bağlamsal olarak iletilmesini kolaylaştırır.

1.6.2 Benzerlik Araması

Cosine similarity metriği kullanılır. FAISS bunu hızlı bir şekilde işler. LlamaIndex ile birleştiğinde, daha gelişmiş sorgularda filtreleme ve reranking gibi yetenekler eklenebilir.

1.6.3 LLM Prompt Yönetimi

Modelin yalnızca belge içeriklerine dayanarak yanıt üretmesi sağlanır. Prompt örneği:

```
Sen bir teknik destek asistanısın. Cevaplarını yalnızca verilen belge parçalarına göre oluştu
Yorum yapma. Yetersiz bilgi varsa bunu belirt.
```

1.7 Gelişmiş Mimariler (LangChain Entegrasyonları)

1.7.1 Query Rewriting (LangChain Chain)

Bazı kullanıcı soruları eksik, kısa veya bağlamdan kopuk olabilir. LangChain ile oluşturulan `query_rewriter_chain.py`, bu tür soruları yeniden yazar ve daha anlamlı hale getirir. Bu da doğru belgelere ulaşma oranını artırır.

1.7.2 Belge Kalite Grading (LangChain + LLM)

`document_grader_chain.py` modülü, getirilen belgelerin gerçekten soruyla ne kadar ilgili olduğunu LLM'e sorarak değerlendirir. Alakasız belgeler çıkarılır, yanıt kalitesi artar.

1.7.3 Fallback Router (LangChain Conditional Chain)

Eğer belge bulunamazsa ya da anlamlı yanıt üretilemezse `fallback_router_chain.py` devreye girer. Alternatif sorgu yazma, yeniden deneme veya bilgi yetersizliği bildirimi gibi seçenekler içerir.

1.7.4 Multi-Query Retrieval (LangChain Rewriting + Merge)

Aynı sorunun farklı biçimlerde yeniden yazılması ve her varyasyon için belge çekilmesi mümkündür. LangChain ile bu varyasyonlar birleştirilir ve daha güçlü sonuçlar elde edilir.

1.8 Kullanıcı Geri Bildirimi

Kullanıcılar, verilen cevabın faydalı olup olmadığını işaretleyebilir. Bu geri bildirimler `feedback_logger.py` ile `logs/queries.jsonl` dosyasına yazılır:

- Soru
- Getirilen belge ID'leri
- Üretilen yanıt
- Geri bildirim (faydalı / değil)
- Zaman damgası

Bu veriler, gelecekte performans analizi ve sistem eğitimi için kullanılabilir.

1.9 Doğruluk ve Değerlendirme

Yöntem	Açıklama
Geri Bildirim	Kullanıcılardan alınan işaretleme ile
Belge Kalite Kontrolü	LangChain zinciri üzerinden LLM ile "evet/hayır" değerlendirme
Kaynak Gösterimi	Her cevapta belge referansı verilir
Query Rewrite Etkisi	Rewrite sonrası daha iyi belgelere ulaşılıyor mu?
Precision@k	İlk `k` dönen belgenin uygunluk oranı manuel olarak ölçülür

1.10 Kullanıcı Akışı

1. Kullanıcı soru sorar
2. Soru normalize edilir ve gerekirse yeniden yazılır
3. LlamaIndex ile embedding yapılır

4. FAISS + LlamaIndex üzerinden belge çağrılır
5. Belge kalitesi LangChain zinciriyle değerlendirilir
6. En uygun belgeler seçilir, modele verilir
7. Model yanıt üretir
8. Kullanıcıya sunulur, kaynak gösterilir
9. Geri bildirim alınır, log dosyasına kaydedilir

1.11 Yol Haritası

1.11.1 MVP Aşaması

1. **Proje Kurulumu ve Veri Seti Hazırlığı**
   Gerekli klasör yapısı oluşturulur, test amaçlı teknik servis belgeleri (`.txt`, `.pdf`, `.md`) `data/` klasörüne eklenir.
2. **Embedding + FAISS + LlamaIndex Kurulumu**
   Belgeler parçalara ayrılır (chunking), `sentence-transformers/all-MiniLM-L6-v2` modeli ile embedding yapılır ve FAISS vektör veritabanına kaydedilir.
3. **Retriever ve LLM Entegrasyonu**
   Kullanıcı sorularını embedding'e çeviren retriever ile OpenRouter API üzerinden LLM'e bağlanan `llm_generator.py` modülü devreye alınır.
4. **Streamlit UI ve Logging**
   Kullanıcı arayüzü geliştirilir, her soru-cevap oturumu `logs/` klasörüne kayıt altına alınır.
5. **Temel Test ve Optimizasyon**
   Test belgeleri üzerinden sistem performansı ölçülür, parametre ayarları (chunk_size, top_k, temperature vb.) optimize edilir.

1.11.2 Gelişmiş Aşama (v2)

1. **Query Rewriting**
   Eksik veya belirsiz soruların LangChain ile yeniden yazılması.

2. **Belge Kalite Grading**
   Getirilen belgelerin LLM ile uygunluk değerlendirmesi ve alakasız belgelerin elenmesi.
3. **Fallback Router**
   Cevap üretilemediğinde alternatif sorgu oluşturma veya kullanıcıya bilgi yetersizliği mesajı verme.
4. **Multi-Query Retrieval**
   Aynı sorunun farklı biçimlerde yazılarak birleştirilmiş sonuçların getirilmesi.
5. **Gelişmiş UI Özellikleri**
   Arama geçmişi görüntüleme, kaynak filtreleme, belge önizleme gibi ek kullanıcı deneyimi geliştirmeleri.
6. **Performans ve Ölçeklenebilirlik İyileştirmeleri**
   Büyük veri setlerinde hız optimizasyonu, gelişmiş indeksleme teknikleri (ör. Raptor Indexing) ve uzun bağlam desteği.

1.12 Gelecek Yol Haritası (v2)

Gelişmiş Özellik	Açıklama
Self-RAG	Geri bildirim sonucu başarısız cevaplarda sorgu yeniden yazılır
Raptor Indexing	Çok sayıda belgeyi hiyerarşik özetleyip indeksleme yöntemi
LangGraph	Tüm RAG adımlarını düğüm-kenar yapısında akıllı kontrol etmek
Web Search Simulation	Belgede yoksa dış kaynak araması simüle etmek (offline)
Belge Özetleme + Long Context	Özet embed edilir, tam belge modele verilir

1.13 Başarı Kriterleri

Kriter	Açıklama
Yanıt doğru mu?	Belgelerle tutarlı, anlamlı içerik üretiliyor mu?

Kriter	Açıklama
Belge referansı var mı?	Hangi kaynağa dayandığı kullanıcıya gösteriliyor mu?
UI kullanımı kolay mı?	Streamlit ile anlaşılır ve hızlı mı?
Geliştirilebilir yapı mı?	LangChain ile genişlemeye uygun mu?
MVP + Gelişmiş akış ayrımı net mi?	MVP sade, v2 planı yol haritasına sahip mi?

1.14 Genel Özet

Bu projede teknik servis süreçlerine yardımcı olan, belge temelli bilgi sağlayan bir yapay zeka asistanı geliştirdim. RAG mimarisi sayesinde sistem sadece cevap üretmekle kalmıyor, cevabın dayandığı belge parçasını da kullanıcıya sunuyor. Gelişmiş senaryolarda LangChain kullanılarak sorgu yazımı, belge puanlama ve fallback gibi üretim kalitesinde bileşenlerle desteklenmiştir. Sistem tamamen açık kaynak, cihaz üstü çalışır ve genişletilmeye uygundur.