🕵️ NLP Proje Görevi: Sektörel ReAct Ajanı Tasarımı

Bu projede, statik cevaplar veren bir Dil Modelini (LLM), düşünen, karar veren ve dış kaynakları kullanarak problem çözen bir **Otonom Ajan'a (ReAct Agent)** dönüştüreceksiniz.

🎯 Projenin Amacı

Size verilen temel kod iskeletini kullanarak; belirlediğiniz özel bir ticari alan için (Örn: E-Ticaret, Hukuk, Sağlık, Lojistik vb.) özelleşmiş bir yapay zeka asistanı geliştirmektir. Ajanınız, o sektöre ait teknik dökümanları veya verileri kullanarak kullanıcının sorularını "muhakeme ederek" (reasoning) cevaplamalıdır.

🔗 Kaynaklar

- **Temel Kod (Notebook):** Google Colab Linki

  https://colab.research.google.com/drive/1P_6jWDjVKEgOgPZXxUCXIeyg5ERIt4Pl?usp=sharing

- **Teorik Kılavuz:** *ReAct - Reasoning and Action.pdf* (Ekte paylaşılmıştır)

📝 Adım Adım Yapılacaklar

1. Adım: Sektör ve İş Modeli Seçimi

Her öğrenci **farklı** bir alan seçmelidir. Seçtiğiniz alanın "teknik bilgi" gerektiren bir yönü olmalıdır.

- *Örnekler:* Sigorta Poliçesi Asistanı, Kripto Para Analisti, İlaç Prospektüs Uzmanı, İmar Yönetmeliği Danışmanı vb.

2. Adım: Veri Toplama ve Bilgi Tabanı Oluşturma (Knowledge Base)

Seçtiğiniz sektöre ait, halka açık (public) verileri toplayın. Bu veriyi modelinize öğretmek için iki yoldan birini seçebilirsiniz:

- **Yol A (RAG - Önerilen):** Verilerinizi "chunk"lara bölün, embeddinglerini çıkarın ve bir Vektör Veritabanına (Chroma/Pinecone vb.) kaydedin.
- **Yol B (LoRa):** Verilerinizle bir Base Modeli (örn: Qwen, Llama, Mistral vs.) fine-tune ederek sektörel bilgiye sahip bir LoRa adaptörü eğitin.

3. Adım: ReAct Mimarisine Geçiş (En Kritik Adım)

Size verilen Colab kodunu kullanarak şu entegrasyonu yapmalısınız:

1. **RAG/LoRa'yı "Tool" Olarak Tanımlayın:**

- Klasik RAG sistemleri soruyu alır ve cevabı verir. ReAct mimarisinde ise sisteminiz cevabı değil, **ham bilgiyi (context)** döndüren bir fonksiyon (Tool) olmalıdır.
- Fonksiyonunuz, Ajanın (LLM) ihtiyaç duyduğunda çağırabileceği bir araç olmalıdır (Örn: insurance_policy_search_tool).

2. **System Prompt Tasarımı:**
   - Ajanınızın beynini tasarlayın. Ona hangi araçlara sahip olduğunu ve nasıl düşünmesi gerektiğini (Thought -> Action -> Observation) anlatan bir prompt yazın.

4. Adım: Senaryo Testleri

Ajanınızı aşağıdaki iki senaryo türünde test edin ve loglarını kaydedin:

- **Senaryo A (Tek Atımlık Sorgu):** Doğrudan dökümandan bulunacak cevaplar.
  - *Örn:* "X sigortasının iptal süresi kaç gündür?"
- **Senaryo B (Çok Adımlı / Multi-Hop Sorgu):** Ajanın önce dökümandan bilgi çekip, sonra bu bilgiyle mantıksal çıkarım yapmasını gerektiren sorular.
  - *Örn:* "Geçen yılın cirosunu rapordan bul ve bugünkü dolar kuruyla TL'ye çevir." (Burada hem rapor okuma hem hesaplama/kur bilgisi gerekir).

📦 **Teslim Edilecekler**

1. **Github Reposu:**
   - Çalışan .ipynb (Colab) dosyası veya Python scriptleri.
   - Kullanılan requirements.txt.
   - Seçilen veri setinden örnekler.
2. **Proje Raporu (PDF):**
   - Seçilen sektör ve nedeni.
   - Kullanılan Yöntem (RAG mi LoRa mı?) ve mimari şeması.
   - **Trace Çıktıları:** Ajanın "Düşünme" adımlarını (Thought, Action, Observation) gösteren örnek loglar.
   - Karşılaşılan zorluklar (Örn: Sonsuz döngüye girme, halüsinasyon vb.) ve çözümleriniz.

💡 **İpuçları ve Uyarılar (Dökümandan Notlar)**

- **Sonsuz Döngü (Infinite Loop):** Ajan bazen cevabı bulamazsa sürekli aynı aramayı yapabilir. Döngü limiti (max_turns) koymayı unutmayın (Örn: En fazla 5 adım).
- **Halüsinasyon:** Ajan elinde olmayan bir aracı (örn: email_sender) uydurup çağırmaya çalışabilir. System Prompt'ta araçlarınızı çok net tanımlayın.
- **Dil Sorunu:** Eğer veriniz İngilizce, sorular Türkçe ise Ajan kafası karışabilir. Prompt içinde "Her zaman Türkçe düşün" komutu vererek bunu çözebilirsiniz.

EK: ReAct (Reasoning + Acting) Mimarisi ve RAG Entegrasyon Rehberi

1. Giriş ve Temel Kavramlar

1.1 ReAct Nedir?

ReAct (**Re**asoning + **Act**ing), Büyük Dil Modellerinin (LLM) sadece metin üreten statik yapılar olmaktan çıkıp, problem çözen otonom ajanlara dönüşmesini sağlayan bir prompt mühendisliği ve mimari paradigmadır.

Yao et al. (2022) tarafından ortaya konan bu yaklaşım, modelin bir görevi yerine getirirken iki temel yeteneği birleştirmesine dayanır:

1. **Reasoning (Muhakeme):** Modelin ne yapacağını planlaması, durumu analiz etmesi ve strateji belirlemesi (Düşünce Zinciri - Chain of Thought).
2. **Acting (Eylem):** Modelin dış dünya ile etkileşime geçmek için belirli araçları (Tools) çağırması (API sorgusu, veritabanı araması, matematiksel işlem vb.).

1.2 ReAct Döngüsü

ReAct ajanı lineer (doğrusal) çalışmaz; döngüsel bir süreç izler:

- **Thought (Düşünce):** "Kullanıcı X'i sordu. Bunu cevaplamak için Y bilgisine ihtiyacım var."
- **Action (Eylem):** "Veritabanında Y'yi ara."
- **Observation (Gözlem):** "Veritabanından gelen sonuç: Z."
- **Thought (Tekrar Düşünce):** "Z bilgisini aldım, şimdi bunu kullanıcının sorusuna uyarlayabilirim."
- **Final Answer (Nihai Cevap):** "Cevap Z'dir."

2. Klasik RAG vs. ReAct Agent Mimarisi

Mevcut sistemlerinizle kıyaslandığında ReAct'ın farkını anlamak, doğru mimariyi kurmak için kritiktir.

2.1 Klasik RAG (Retrieval-Augmented Generation)

Klasik RAG akışı "Tek Atımlık" (One-Shot) bir süreçtir ve deterministiktir.

- **Akış:** Sorgu -> Embedding -> Retrieval -> Context Birleştirme -> LLM Cevap Üretimi.
- **Eksiklik:** Eğer çekilen döküman soruyu cevaplamaya yetmezse, sistem tıkanır veya halüsinasyon görür. Modelin "Bu bilgi yetersiz, başka bir yere bakmalıyım" deme şansı yoktur.
- **Rol:** LLM burada sadece bir "Okuyucu/Özetleyici"dir.

2.2 ReAct Agentic RAG

ReAct yapısında RAG, sistemin tamamı değil, sadece **bir aracıdır (Tool).**

- **Akış:** Sorgu -> Ajan (Reasoning) -> Karar -> **RAG Tool (Retrieval)** -> Ajan (Observation) -> Ajan (Cevap).
- **Avantaj:** Ajan, RAG'dan gelen bilgiyi analiz eder. Eğer bilgi eksikse aramayı tekrarlar (Self-Correction) veya farklı bir araç kullanır (Tool Switching).
- **Rol:** LLM burada bir "Orkestratör/Karar Verici"dir.

3. Mimari Konumlandırma ve Entegrasyon

Mevcut RAG pipeline'ınızı ReAct mimarisine entegre ederken bileşenlerin rolleri yeniden dağıtılmalıdır.

3.1 Katmanlı Mimari

1. **Orkestrasyon Katmanı (The Brain):**
   a. ReAct Agent burada çalışır.
   b. System Prompt burada tanımlanır.
   c. Hafıza (Conversation History) burada tutulur.
2. **Araç Katmanı (The Limbs):**
   a. RAG Tool: Vektör veritabanına erişim sağlar.
   b. Math Tool: Hesaplamalar yapar.
   c. API Tool: Dış veri (Hava durumu, borsa vb.) çeker.
3. **Veri Katmanı (The Knowledge):**
   a. Vector DB (Chroma, Pinecone vb.)
   b. Dökümanlar (PDF, TXT)

3.2 RAG Bileşenlerinin Dönüşümü

Mevcut RAG yapınızdaki bileşenlerin ReAct'a adaptasyonu:

Bileşen	**Klasik RAG Rolü**	**ReAct Mimarisi Rolü**	**Yapılması Gereken Değişiklik**
Chunking & Embedding	Veri Hazırlığı	Veri Hazırlığı	**Değişiklik Yok.** Aynı altyapı korunur.
Vector Database	Ana Bilgi Kaynağı	Ana Bilgi Kaynağı	**Değişiklik Yok.**
Retriever	Döküman Getirici	Döküman Getirici	Bir Python fonksiyonuna (Tool) sarılır.

LLM (Generation)	Cevap Üretici	**Kaldırılır**	RAG fonksiyonu LLM cevabı değil, **ham metin (Raw Context)** döndürmelidir.
System Prompt	Cevaplayıcı	**Yönlendirici**	Prompt artık cevabı değil, "Ne zaman RAG'a gidileceğini" tarif eder.

4. Uygulama Adımları: Minimum Refactor ile Geçiş

Öğrencilerin projelerini bozmadan ReAct'a geçiş yapmaları için izlemeleri gereken teknik yol haritası:

Adım 1: RAG Pipeline'ını "Sessizleştirmek"

Mevcut RAG kodunuz muhtemelen retrieve edip ardından generate etmektedir.
generate kısmını (yani LLM'e giden son adımı) kesin.

- **Eski Çıktı:** "Şirket politikasına göre izinler 14 gündür." (LLM Cümlesi)
- **Yeni Çıktı:** "Döküman ID: 102 - İçerik: ...izinler ilk yıl 14 gündür..." (Ham String)

Adım 2: RAG Tool Wrapper Yazımı

Retriever fonksiyonunu, Ajanın kullanabileceği bir formata sokun.

• Pseudo-Code (Kavramsal Kod):

Python def rag_knowledge_base_tool(sorgu):
"""
Şirket içi dökümanlarda, kurallarda ve raporlarda arama yapar.
Genel sohbet için kullanma.
""" belgeler = vector_db.similarity_search(sorgu) return birlestir(belgeler) # Sadece metin döndür

Adım 3: ReAct Prompt Tasarımı

Ajanın beynini (System Prompt) yapılandırın.

• Prompt Şablonu:

Sen bir ReAct ajanısın. Aşağıdaki araçlara erişimin var:
1. rag_knowledge_base_tool: Sadece dökümanlardan bilgi gerektiren sorularda kullan.
Düşün (Thought) -> Aksiyon (Action) -> Gözlem (Observation) döngüsünü takip et.
Asla kendi bilgine güvenme, mutlaka RAG aracını kontrol et.

5. Örnek Akış Senaryoları (Trace)

ReAct + RAG entegrasyonunun gücünü gösteren iki farklı senaryo. **Senaryo A: Doğrudan RAG Sorgusu**

Kullanıcı: "Staj defterini ne zaman teslim etmeliyim?"

1. **Agent (Thought):** Kullanıcı staj prosedürünü soruyor. Bu bilgi şirket dökümanlarında olmalı. rag_knowledge_base_tool kullanmalıyım.
2. **Agent (Action):** rag_knowledge_base_tool("staj defteri teslim tarihi")
3. **System (Execution):** Vektör DB taranır, ilgili paragraf bulunur.
4. **Agent (Observation):** "Yönetmelik Madde 5: Staj defterleri, dönemin son gününden itibaren 7 iş günü içinde teslim edilir."

5. **Agent (Thought):** Bilgiyi buldum. Şimdi kullanıcıya cevap verebilirim.
6. **Agent (Final Answer):** "Staj defterinizi dönem bitiminden sonraki 7 iş günü içinde teslim etmelisiniz."

Senaryo B: Multi-Hop (Çok Adımlı) Sorgu

Kullanıcı: "Geçen yılın faaliyet raporundaki net kârım ile bugünkü Apple hisse fiyatını karşılaştır."

1. **Agent (Thought):** İki bilgiye ihtiyacım var. 1) Rapordaki net kâr (RAG), 2) Güncel Apple hissesi (Dış API). Önce raporu bulayım.
2. **Agent (Action):** rag_knowledge_base_tool("2023 faaliyet raporu net kâr")
3. **Agent (Observation):** "...2023 yılı net kârı 150 Milyon USD olarak gerçekleşmiştir..."
4. **Agent (Thought):** Kârı öğrendim (150M). Şimdi Apple hissesini öğrenmeliyim.
5. **Agent (Action):** finance_tool("AAPL price")
6. **Agent (Observation):** "220 USD"
7. **Agent (Thought):** Şimdi karşılaştırma yapabilirim (Bu örnekte elmalarla armutlar olsa da, ajan isteneni yapar).
8. **Agent (Final Answer):** "2023 net kârınız 150 Milyon USD iken, Apple hissesi şu an 220 USD'dir."

6. Sık Karşılaşılan Hatalar ve Öneriler

1. **Sonsuz Döngü (Infinite Loop):** Ajan RAG'dan dönen cevabı beğenmezse sürekli aynı aramayı yapabilir.
    a. *Çözüm:* Döngü limitini (max_turns) 3-5 arasında tutun.
2. **Tool Halüsinasyonu:** Ajan var olmayan bir aracı (örn: email_sender) çağırmaya çalışabilir.
    a. *Çözüm:* System Prompt içinde araç listesini çok net belirtin.
3. **Dil Karmaşası:** RAG'dan Türkçe veri gelip, Ajan İngilizce düşünebilir.
    a. *Çözüm:* System Prompt'ta "Her zaman Türkçe düşün ve cevap ver" talimatı verin veya RAG verisini özetleterek alın.