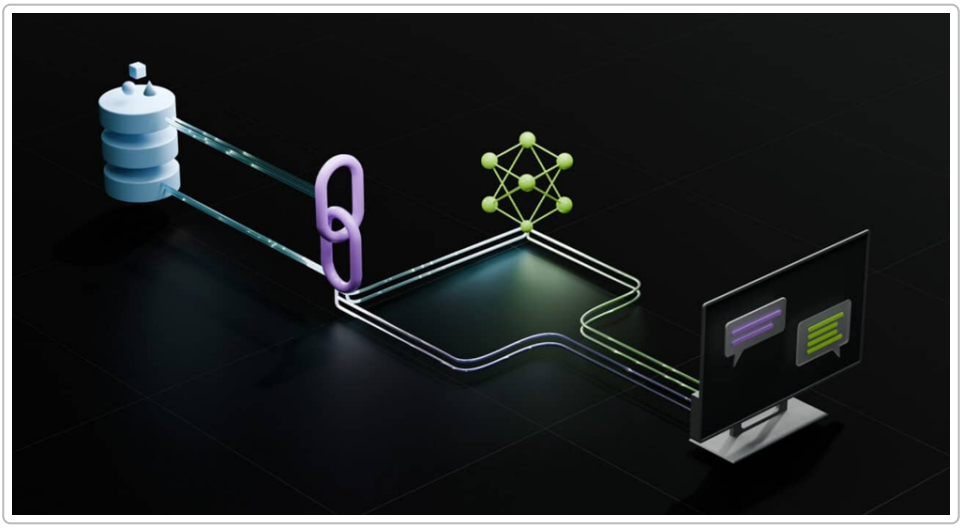
ChatGPT

RAG, Word Embeddings, LangChain, Chroma ve Semantik Analiz: Son 5 Yıldaki Gelişmeler

1. RAG (Retrieval-Augmented Generation) Mimarisi



Şekil 1: RAG yaklaşımında, büyük dil modeli bir veri deposundan alınan bilgileri zincirleme şekilde kullanarak kullanıcıya yanıt üretir (temsilî görsel).

Retrieval-Augmented Generation (RAG), büyük dil modellerinin dış bilgi kaynaklarından yararlanarak daha **doğru ve güvenilir** yanıtlar üretmesini sağlayan bir mimaridir [1]. 2019-2020 yıllarında Facebook AI (Meta) araştırmacıları tarafından ortaya atılan bu yaklaşım, büyük dil modellerinin sadece kendi parametrelerine gömülü bilgiyle sınırlı kalmasını aşmayı hedefler [2]. RAG sistemi iki temel bileşenden oluşur: (i) **Bilgi getirme (retrieval)** aşaması – burada model, harici veri tabanlarından veya belgelerden kullanıcı sorgusuyla ilgili kesitleri arar ve getirir; (ii) **Cevap üretme (generation)** aşaması – burada büyük dil modeli, kendi dil bilgisi yeteneğini getirdiği bu harici bilgilerle birleştirerek yanıt üretir [3]. Bu sayede RAG, bir LLM'i "kapalı kitap sınavı" yerine **"açık kitap sınavı"** modunda çalıştırmaya benzer: model, yanıt verirken sadece parametrelerine kazınmış hafızasına değil, aynı zamanda bir **"kitap"** misali harici bilgi deposuna bakarak cevaplar üretir [4].

RAG mimarisi, özellikle **bilgi yoğun (knowledge-intensive)** görevlerde büyük avantaj sağlamıştır. Model, harici kaynaktan gelen gerçek bilgilerle **temellendirildiği (grounding)** için hem en güncel ve doğru bilgilere erişebilir, hem de kullanıcıya cevabın **kaynağını referans göstererek** güven verir [5] [6]. Örneğin, bir RAG tabanlı soru-cevap sistemi hem güncel ansiklopedik bilgileri kullanabilir hem de cevabın alındığı belgeye atıf yaparak kullanıcıya şeffaflık sunar. Bu yöntem, LLM'lerin sıkça yaşadığı halüsinasyon (uydurma bilgi) problemini azaltır ve cevapların doğrulanabilir olmasını sağlar [7]. Ayrıca model parametrelerini güncellemeden, sadece bilgi kaynağını güncelleyerek sistemi beslemek mümkün olduğu için, **sürekli yeniden eğitime duyulan ihtiyacı azaltır** ve maliyetleri düşürür [8].

Son 5 yılda RAG alanında önemli gelişmeler yaşanmıştır. 2020'de Patrick Lewis ve ekibinin öncü makalesi, RAG'ı *"hemen hemen her LLM'i herhangi bir harici bilgi kaynağına bağlayabilen genel amaçlı bir*

ince ayar tarifi" olarak tanımladı [9]. İlk dönem "naif RAG" yaklaşımında model, tek bir adımdaki sorgu için ilgili belgeleri getirip cevabı üretirken; daha **ileri RAG** tekniklerinde iteratif arama, geri bildirimle iyileştirme (örn. *self-RAG* gibi yaklaşımlar) ve modüler yapılar denenmiştir [10]. Örneğin, modelin önce kendi ürettiği cevabı kontrol edip eksik veya yanlış kısımlar için tekrar belge getirmesi gibi **öz-düzeltmeli RAG** yöntemleri literatürde yer almaya başlamıştır. RAG bileşenlerinde de ilerlemeler kaydedilmiştir: **getirici kısımda** klasik TF-IDF veya BM25 yerine yoğun vektör arama (embedding tabanlı arama) ve öğrenilebilir retriever modelleri kullanılmış; **üretici kısımda** ise daha büyük ve yetenekli LLM'ler, daha uzun bağlam pencere desteğiyle entegre edilmiştir [11]. Hatta bazı sistemler, **arttırma (augmentation)** teknikleriyle getirilen metinleri modelin parametrelerine geçici olarak enjekte etmeyi (örneğin kilit bilgileri prompt içinde işaretlemeyi) deneyerek LLM'in bu bilgilere azami dikkati vermesini sağlamıştır [12].

Kullanım alanları: RAG mimarisi, güncel olarak çok çeşitli uygulamalarda benimsenmiştir. Özellikle **açık uçlu soru-cevap** ve **sohbet botları** RAG'den büyük fayda görmektedir – model, kullanıcı sorusuna yanıt vermeden önce ilgili dokümanlardan arka plan bilgisini çeker ve cevabını bu bilgiyle temellendirir. **Kurumsal uygulamalarda**, şirketler kendi dokümanlarını, politikalarını veya veri tabanlarını bir *bilgi kaynağı* haline getirip *"kendi verim için ChatGPT"* benzeri asistanlar geliştirmektedir [13] [14]. Örneğin, tıbbi bir dil modeli, tıp literatüründen veya ilaç kılavuzlarından arama yaparak doktorlara destek olabilir; finansal bir model, piyasa verilerine erişerek analistlere yardımcı olabilir [15]. Mühendislik, hukuk, eğitim gibi alanlarda da LLM'lerin uzman belgelerle beslenip doğru yanıtlar vermesi için RAG yaklaşımı kullanılmaktadır. Bunun yanı sıra RAG, **müşteri hizmetleri** (SSS dokümanlarını tarayıp yanıt verme), **eğitim** (içerik tarayıp öğrenci sorularını yanıtlama) ve **iş zekâsı** (raporlar içinde arama yaparak özet çıkarma) gibi pek çok senaryoda uygulanmaktadır. Büyük teknoloji şirketleri ve girişimler de RAG'ı hızla benimsemiştir: AWS, IBM, Glean, Google, Microsoft, NVIDIA, Oracle gibi firmalar son yıllarda RAG tabanlı hizmetler veya araçlar duyurmuştur [16]. Örneğin, Microsoft'un Copilot ve benzeri LLM tabanlı asistanlarında, kullanıcının kurumsal verilerine erişim için RAG bileşenleri bulunur. NVIDIA ise 2023'te **NeMo** çerçevesine RAG desteği ekleyerek kurumsal veriler üzerinde sohbet botları oluşturmayı kolaylaştıran bir *Blueprint* yayımlamıştır [17].

Özetle, RAG mimarisi son 5 yılda **LLM ekosisteminin vazgeçilmez bir parçası** haline gelmiştir. LLM'lerin **sınırlı bilgi güncelliği** ve **halüsinasyon** problemlerine pratik bir çözüm sunan bu yaklaşım, araştırmadan endüstriye geniş kabul görmüştür. Güncel çalışmalar, RAG sistemlerinin daha da geliştirilerek çok modlu bilgilere erişim, etkileşimli sorgu-cevap döngüleri ve daha verimli getirme-üretme stratejileri üzerine yoğunlaşmaktadır [18]. RAG, büyük dil modellerini gerçek dünya bilgisinin geniş denizine bağlayan bir köprü olarak, **doğal dil işleme uygulamalarının güvenilirliğini ve becerisini arttıran** kritik bir mimari olarak konumlanmıştır.

2. Türkçe için Word Embeddings

Word embedding yöntemleri, Türkçe doğal dil işleme alanında son yıllarda önemli ilerlemeler kaydetmiştir. Türkçe, eklemeli (aglutinatif) bir dil olduğu için kelime gömme modellerinin bu dile uyarlanması bazı özel zorluklar içerir. Son 5 yılda, hem **Türkçeye özgü** (tek dilli) gömme modelleri geliştirilmiş, hem de **çok dilli** büyük modeller içerisinde Türkçe desteği iyileştirilmiştir. Bu bölümde, öne çıkan embedding yöntemleri ve modellerini, performans değerlendirmelerini ve güncel durumlarını özetliyoruz.

- **Statik Kelime Gömme (Word2Vec, GloVe, FastText):** İlk gömme teknikleri olan Word2Vec ve GloVe, 2010'ların ortalarında Türkçe için de uygulanmıştır. Word2Vec'in Süreç-Gram (Skip-gram) ve ÇBKT (CBOW) gibi varyantları Türkçe büyük metinlerde eğitilerek kelimeleri vektör uzayında temsil etmiştir. Araştırmalar, Word2Vec'in Türkçe üzerindeki performansının GloVe'e kıyasla daha

iyi olabildiğini ve eğitim hızının daha yüksek olduğunu göstermiştir [19]. Örneğin, Şen ve Erdoğan (2014), bir Türkçe korpus üzerinde Skip-gram modeliyle 300 boyutlu vektörlerin kelime anlamsal görevlerinde en iyi sonucu verdiğini bulmuştur. FastText modeli ise kelimeleri karakter n-gram'larına bölerek temsil ettiği için, Türkçedeki çekim ekleri ve türetme ekleri gibi morfolojik yapıları daha iyi yakalayabilmektedir. Dündar & Alpaydın (2019) üç farklı Türkçe veri setinde Word2Vec, FastText ve ELMo'yu kıyaslamış; sonuçta **FastText'in isim ve fiil çekimlerini daha iyi temsil ettiğini**, buna karşın Word2Vec'in özellikle anlamsal benzerlik (ör. benzerlik analogileri) görevlerinde diğerlerini geçtiğini rapor etmiştir [20]. Hatta bazı sınıflandırma görevlerinde basit *bag-of-words* modellerinin bile gömülü model kullanan derin modelleri yakalayabildiği belirtilmiştir [21]. Bununla birlikte, gömülü kelime temsilleri derin öğrenme modelleriyle birlikte kullanıldığında bariz faydalar sağlar: Örneğin bir çalışma, LSTM tabanlı bir metin sınıflandırma sistemine FastText vektörleri verildiğinde Türkçe metin sınıflandırma başarımının belirgin şekilde arttığını göstermiştir [22].

- **Bağlamsal (Contextual) Gömme Modelleri:** 2018 sonrası dönemde, BERT ve türevi *transformer* tabanlı modeller, kelime gömme yaklaşımını kökten değiştirmiştir. Artık her kelime, cümle bağlamına göre farklı bir vektör alabilmektedir (bağlamsal embedding). Türkçe için ilk büyük adımlardan biri, **multilingual BERT (mBERT)** modelinin duyurulmasıydı – bu model 100'den fazla dil ile birlikte Türkçeyi de kapsıyordu. Ancak çok dilli modelin kapasitesi tüm dillere paylaştırıldığı için, belirli görevlerde tek dillilere kıyasla optimal değildi. 2020'de **BERTurk** adıyla bilinen Türkçe BERT modeli yayınlandı (Stefan Schweter tarafından) [23]. BERTurk, tamamen Türkçe Wikipedia ve kapsamlı bir yerli metin korpusu üzerinde eğitilmiş, *BERT-Base* mimarisinde bir modeldir. Bu model, Türkçe dil anlama görevlerinde (örn. isim tanıma, duygu analizi, okuma anlama) o döneme kadarki en yüksek sonuçları elde etmiştir. Benzer şekilde, yine 2020'lerde bazı araştırma grupları **Electra** ve **RoBERTa** mimarilerini Türkçe için uyarlayarak tek dilli modeller geliştirmiştir (örn. BERTurk'e benzer şekilde Türkçe RoBERTa deneyleri). Öte yandan, Facebook AI tarafından yayınlanan **XLM-RoBERTa (XLM-R)** modeli (2019) çok dilli olmasına rağmen geniş veri ve gelişmiş eğitim sayesinde Türkçe de dahil birçok dilde çok iyi performans göstermiştir. Türkçe dilbilgisi yapısını öğrenmede güçlü olan XLM-R, özellikle ince ayar yapıldığında, bazı Türkçe görevlerde BERTurk seviyesinde sonuçlar verebilmektedir.

- **ELMo ve Diğer Yöntemler:** BERT öncesi bağlamsal yöntemlerden **ELMo** (2018), Türkçe için adaptasyonu yapılan ilk derin bağlamsal modeldir. Che et al. (2018) çalışması kapsamında Türkçe için ELMo gömüleri üretilmiş, böylece her kelime için hem sol hem sağ bağlamı hesaba katan vektör temsilleri elde edilmiştir [24]. ELMo Türkçe görevlerde Word2Vec/fastText gibi statik modellere kıyasla daha yüksek doğruluk sağlamış ancak BERT'in gelişiyle gölgede kalmıştır. Güncel olarak, **Sentence-BERT** gibi cümle düzeyinde gömme sağlayan modeller de Türkçeye uyarlanmıştır; örneğin LaBSE (Language-agnostic BERT Sentence Embedding) ve benzeri çok dilli cümle embedding modelleri Türkçe cümleler için anlamsal karşılaştırma, arama görevlerinde kullanılmaktadır.

- **Performans Değerlendirmeleri:** Son yıllardaki çalışmalar, **modern modellerin bariz üstünlüğünü** ortaya koyuyor. 2023 yılında Bayrak ve arkadaşlarının bir çalışmasında, Türkçe metin sınıflandırma görevinde Word2Vec, FastText, ELMo ve BERT tabanlı gömüler karşılaştırılmış; **BERT tabanlı temsil** en yüksek başarıma ulaşarak en başarılı model olmuştur [25]. Benzer şekilde, Arslan vd. (2023) Türkçe Word2Vec, GloVe ve FastText modellerini çeşitli içsel testlerle değerlendirip kapsamlı bir karşılaştırma sunmuştur [26]. 2024 yılında yayımlanan kapsamlı bir çalışma ise statik gömüler konusunda yeni bir perspektif getirmiştir: Bu araştırmada BERT ve ELMo gibi *bağlamsal modellerden* çıkarılan **statik kelime vektörlerinin**, klasik statik modelleri bir çok görevde geride bıraktığı gösterilmiştir [27]. Özellikle X2Static adı verilen yöntemle BERT'in her kelime için sabit bir vektör elde edecek şekilde damıtılması sonucunda,

analogi testleri, anlamsal benzerlik ve metin sınıflandırma gibi görevlerde bu vektörler Word2Vec/FastText gibi modellere üstünlük sağlamıştır [27]. Bu sonuç, sınırlı hesaplama kaynağı olan durumlarda bile büyük modellere dayalı embed'lerin gücünden faydalanılabileceğini göstermektedir. Özetle, Türkçe NLP uygulamalarında **FastText** gibi karakter seviye modeller morfolojik esneklik sağlarken, **BERT tabanlı modeller** genel anlamda en yüksek doğruluğu sunmaktadır. Mevcut en iyi yaklaşımlardan biri, mümkünse BERT-tabanlı bir modeli doğrudan kullanmak; eğer kaynak kısıtlıysa onun çıkardığı embedding'leri kullanarak daha hafif modellerle çalışmaktır.

- **Açık Kaynak Modeller ve Kaynaklar:** Türkçe için pek çok açık kaynak embedding modeli ve aracı bulunmaktadır. Örneğin, Ali Köksal tarafından derlenen **Turkish Word2Vec** deposu, geniş bir Türkçe korpus üzerinde eğitilmiş Word2Vec vektörlerini sağlar [23]. **inzva** topluluğu tarafından paylaşılmış **Turkish GloVe** modeli, Wikipedia ve diğer metinlerden elde edilen GloVe vektörlerini sunmaktadır [23]. **BERTurk** modeli ve ağırlıkları HuggingFace aracılığıyla da erişilebilir durumdadır [24]. Facebook AI'nın yayınladığı **fastText** ön-eğitimli vektörleri içinde Türkçe de bulunmaktadır (2 milyon Türkçe kelime vektörü, 300 boyutlu) [24]. Keza AllenNLP'nin çok dilli **ELMo** ağırlıklarında Türkçe yer almaktadır [28]. Son yıllarda bu modelleri derleyip tek bir platformda sunma girişimleri de olmuştur: Örneğin Boğaziçi Üniversitesi'nden bir grup, Türkçe için farklı gömme modellerini içeren bir depo ve değerlendirme kıyaslama setleri hazırlamıştır [29] [30]. Bu sayede araştırmacılar ve geliştiriciler, ihtiyaç duydukları model türünü (statik veya bağlamsal) ve ilgili ön-eğitimli vektörleri kolayca bulabilmektedir.

Güncel Modeller: 2023 itibariyle Türkçe dil işleme için en güncel ve başarılı modeller arasında **BERTurk**'ün devamı niteliğindeki büyük dil modelleri, örneğin daha büyük veriyle eğitilmiş versiyonlar veya Türkçe-de hem metin hem kod gibi çoklu becerilere sahip modeller yer alıyor. Ayrıca, **GPT-3/4 gibi devasa modellerin** çok dilli sürümleri (OpenAI, Google PaLM 2, vb.) Türkçe'yi desteklemekte ve yüksek kalitede çıktı verebilmektedir. Ancak bu modeller kamuya açık ağırlıklar sunmadığından, açık kaynak topluluğu için hala BERT-tabanlı veya XLM-R tabanlı modeller en güçlü seçeneklerdir. 2024 yılında yayınlanan **LLaMA 2** gibi büyük açık modellerin de Türkçe performansı dikkat çekicidir (örneğin, 7 milyar parametreli LLaMA 2, sıfır-shot sorularda Türkçe makul çıktılar verebilmektedir). Yine de, özel bir görev için ince ayar yapılacaksa, Türkçe monolingual modellerin performans ve verimlilik avantajı sürmektedir. Sonuç olarak, **Türkçe NLP alanında word embedding teknolojileri**, Word2Vec'ten BERT'e uzanan çizgide ciddi bir evrim geçirmiş ve günümüzde hemen her uygulama için uygun bir açık kaynak embedding modeli bulunabilir hale gelmiştir.

3. LangChain Framework'ü

LangChain, büyük dil modeli tabanlı uygulamalar geliştirmek için ortaya çıkan ve son iki yıl içinde hızla popülerleşen bir açık kaynak **framework**'tür. 2023 yılında Harrison Chase tarafından başlatılan bu proje, LLM'lerin yeteneklerini *"zincirleme"* şekilde birleştirmeyi, harici veri kaynakları ve araçlarla entegre etmeyi kolaylaştıran bir altyapı sunmasıyla büyük ilgi gördü. LangChain'in temel amacı, **LLM tabanlı uygulamaların geliştirme yaşam döngüsünü basitleştirmek** ve geliştiricilere esnek bir platform sağlamaktır [31].

Öne çıkan özellikleri: LangChain, LLM uygulamaları kurarken ihtiyaç duyulan bir dizi bileşeni kutudan çıkmış şekilde sunar. Bunların başında **zincirler (chains)** gelir – bir zincir, belirli bir mantık sırasıyla LLM çağrılarını, istemci girdi/çıktı işlemlerini ve diğer işlemleri birbirine bağlar. Örneğin bir *"özetleme zinciri"*, önce bir metin getirme fonksiyonunu çağırıp ardından özetleme için LLM'i çalıştırabilir. İkinci önemli bileşen **ajanlar (agents)** ve **araçlar (tools)**dır. LangChain, LLM'lerin birer otonom ajan gibi davranıp belirli *araçları kullanabilmesini* sağlar (arama motoru, hesap makinesi, veri tabanı sorgulayıcı vb. gibi)

[32]. Örneğin bir LangChain ajanı, kullanıcının sorusunu anlamlandırdıktan sonra bir arama aracını çağırıp bulduğu sonuçları tekrar LLM'e vererek nihai cevabı oluşturabilir. Bu sayede çok adımlı görevlerde LLM'in **plan yapma ve yürütme (plan-and-execute)** kabiliyeti ortaya çıkar; hatta insan müdahalesi olmadan ardışık eylemlerle bir hedefe ulaşmaya çalışabilir [33].

LangChain'in bir diğer kilit özelliği **hafıza (memory)** yönetimidir. Varsayılan olarak LLM API'leri (ör. OpenAI ChatGPT API) *durumsuzdur*; yani geçmiş konuşmaları veya işlem adımlarını hatırlamaz, her seferinde tüm geçmişi yeniden vermek gerekir. LangChain, konuşma geçmişini veya durum bilgisini saklamak için dahili bir hafıza arayüzü sunar [34]. Bu arayüz, geliştiricinin seçimine göre farklı veri depolarını kullanabilir: *yerel liste, dosya*, veya özel bir **vektör veritabanı** gibi. Nitekim LangChain, sohbet geçmişini embedding'ler şeklinde kaydedip gerektiğinde çağırmak üzere halihazırda **10'dan fazla vektör veritabanı ile entegre olabilir** (ör. Chroma, Pinecone, Weaviate, FAISS vb.) [35]. Bu sayede, bir sohbet botu uygulaması geliştirirken diyaloğun uzun geçmişini yönetmek, önemli kısımları unutmamak veya önemsiz detayları atmak gibi işlemler LangChain ile oldukça kolay hale gelir. Framework, geliştiriciye sadece hangi hafıza deposunu kullanacağını ve kaç mesajı tutacağını belirtme imkânı verir; gerisini kendi halleder.

Neden ihtiyaç duyuldu?: LLM'lerle çalışmak, ortaya çıktıkları ilk dönemde bir dizi "köşe durum" sorunuyla gelmişti. **Prompt tasarımı, zincirleme istekler, çıktıları izleme**, hata ayıklama, farklı LLM servislerini deneme gibi işler için elle kod yazmak zahmetliydi. LangChain bu zorlukları adreslemek üzere ortaya çıktı. Logan Kilpatrick'in bir değerlendirmesinde belirtildiği gibi, LangChain geliştiricilere **prompt zincirleme, loglama, callback mekanizmaları, kalıcı hafıza yönetimi ve çoklu veri kaynaklarına kolay bağlanma** gibi işlevleri kutudan çıktığı haliyle sunar [36]. Örneğin, normalde bir LLM'e birkaç aşamalı talimat verip yanıtını işleyerek bir sonraki talimatı hazırlamak (chaining) manuel string işlemleri gerektirebilirken, LangChain bunun için hazır *Chain* sınıfları ve *PromptTemplate* araçları sağlamaktadır. Yine farklı LLM sağlayıcılarını (OpenAI, Cohere, AI21, HuggingFace vs.) tek bir ara yüzden çağırmak, herhangi birini kolayca değiştirebilmek LangChain ile mümkündür [37]. Bu *model-agnostik* yapısı sayesinde, bir şirket aynı kod altyapısını kullanarak çeşitli dil modeli API'lerini test edebilir, en iyi sonucu vereni seçebilir [37]. Kısacası LangChain, LLM uygulama geliştirmenin etrafındaki *pürüzlü noktaları yumuşatan* bir araç seti olarak ortaya çıkmıştır.

Topluluk ve Yaygınlık: LangChain açık kaynak projesi etrafında kısa sürede devasa bir geliştirici topluluğu oluşmuştur. Proje GitHub üzerinde 2023 ortasında 50 bin yıldız sayısını geçmiş, aylık indirme sayısı milyonu bulmuş ve yüzlerce kişi projeye katkı vermiştir [38]. Hatta 2023 Haziran'ında 1.000'den fazla farklı kişinin LangChain deposuna katkıda bulunduğu rapor edilmiştir ki bu açık kaynak dünyasında nadir görülen bir ivmedir [39]. Bu büyük ilgi, projenin canlı bir Discord topluluğu, düzenli blog yazıları ve örnek uygulamalarla desteklenmesini de beraberinde getirmiştir. 2024 yılına gelindiğinde LangChain, GenAI (üretken yapay zeka) alanında *"en büyük geliştirici topluluğuna"* sahip projelerden biri haline gelmiştir; resmi web sitesi verilerine göre 1 milyondan fazla kişi bu framework'ü kullanmaktadır [40]. Ayrıca Python dilinin yanı sıra JavaScript/TypeScript sürümü de geliştirilmiş, böylece tarayıcı tabanlı veya Node.js ortamlarında da LangChain kullanılabilir olmuştur.

Kullanım Senaryoları: LangChain genellikle **LLM destekli chatbot** ve **soru-cevap sistemleri** kurmak için tercih edilmektedir. Örneğin, bir şirketin dökümanlarını indeksleyip soruları yanıtlama uygulaması (RAG tabanlı bir bot) LangChain ile birkaç düzine satır kodla prototiplenebilir. Ayrıca **çok adımlı işlem otomasyonu** (ör. bir soruya önce arama yap sonra hesaplama yap gibi), **içerik üretimi iş akışları** (belirli bir tarzda metin üretip sonra bunu başka bir modele değerlendirtme vb.), **agent** tabanlı görev otomasyonu (LLM'e internette arama yaptırıp bulduğu bilgiyle karar verdirtme gibi) alanlarında da LangChain yoğun biçimde kullanılır. 2023'te popüler olan *AutoGPT* gibi deneysel projeler dahi LangChain altyapısını kısmen kullanarak ajanın araç kullanımını ve görev yönetimini sağlamışlardır. Bunların yanında LangChain; **deneme-yanılma ile prompt iyileştirme**, **farklı modellerin çıktısını**

karşılaştırma (A/B test) gibi geliştirici işlerini kolaylaştıran modüller de içerir [41]. Örneğin *ComparisonChain* ile aynı girdiye karşı iki farklı modeli çalıştırıp sonuçlarını kıyaslamak mümkündür.

Ekosistem ve Entegrasyonlar: LangChain zengin bir eklenti ekosistemine sahiptir. Veritabanlarından web hizmetlerine, bulut depolama araçlarından PDF okuma kütüphanelerine kadar >**600 entegrasyon** desteklediği belirtilmektedir [42]. Özellikle vektör veritabanı entegrasyonları (Chroma, Pinecone, Weaviate, Qdrant vb.) ve model entegrasyonları (OpenAI, HuggingFace, Azure AI, Cohere vs.) çok değerlidir. Bu sayede geliştirici, mevcut veri altyapısını veya tercih ettiği ML hizmetini kolayca LangChain zincirine dahil edebilir. Örneğin, bir **LangChain + Chroma** tabanlı RAG pipeline'ında, `RetrievalQA` zinciri kullanılarak belge arama ve yanıt üretme işlemi birkaç satır kodla kurulabilir. LangChain aynı zamanda **LangSmith** adlı bir izleme/ayar hizmeti, **LangChain Hub** adlı bir prompt/model paylaşım platformu gibi yan ürünlerle ekosistemini genişletmiştir. Bu canlı ekosistem, framework'ün güncel kalmasına ve endüstride kabul görmesine katkı sağlamaktadır.

Özetle, LangChain framework'ü, **LLM uygulamalarını prototipleme ve üretime geçirme konusunda adeta bir "standart" haline gelmiştir**. Pek çok başlangıç seviyesi geliştirici için öğrenme eğrisini düşürmüş, ileri seviye kullanıcılar için ise modüler ve genişletilebilir yapısıyla esneklik sunmuştur. Her ne kadar son dönemde bazı geliştiriciler "aşırı soyutlama" veya performans kaygıları nedeniyle eleştiriler getirse de, LangChain'in getirdiği kavramsal çerçeve (zincirler, ajanlar, hafıza vb.) kalıcı olacağa benzemektedir. İlerleyen yıllarda, temel LLM arayüzlerinin gelişmesiyle belki framework'e ihtiyaç azalabilir; ancak platform-bağımsız ve birleştirici bir katman olarak LangChain, büyük dil modellerinin gerçek dünya uygulamalarına entegrasyonunda önemli bir rol oynamaya devam edecektir [43].

4. Chroma Vektör Veritabanı

Chroma (ChromaDB), son yıllarda popülerlik kazanan açık kaynaklı bir **vektör veritabanıdır**. Özellikle *LLM* uygulamalarında **embedding**'leri depolamak ve hızlı benzerlik aramaları yapmak için tasarlanmıştır [44]. 2022'nin sonunda ilk sürümü çıkan Chroma, 2023 boyunca geliştiriciler arasında hızla yayıldı ve RAG gibi uygulamaların **altyapı parçası** haline geldi [45]. Chroma'nın başlıca görevi, metin veya diğer medya temsillerine ait yüksek boyutlu vektörleri veritabanında tutmak ve verilen bir sorgu vektörüne en benzer kayıtları çok hızlı bir şekilde getirmektir.

Teknik Mimari: Chroma'nın iç yapısı, **yakın komşu arama (Nearest Neighbor Search)** problemine optimize edilmiştir. Milyonlarca vektör arasında arama yaparken doğrusal tarama (brute-force) çok yavaş olacağı için, Chroma **yaklaşık yakın komşu arama** yöntemlerini kullanır. Özellikle, **HNSW (Hierarchical Navigable Small World)** adlı graf tabanlı algoritmayı uygulamaktadır [46] [47]. HNSW, vektörleri çok katmanlı bir graf yapısında organize ederek arama süresini büyük ölçüde hızlandırır: Üst katmanlarda seyrek bir bağlantı ağı kurulup hızla yaklaşık konuma gidilir, alt katmanlarda daha ince detaylarla arama keskinleştirilir. Bu sayede pratikte arama karmaşıklığı ~O(log N) seviyesine inebilir [48]. Chroma, HNSW yapılandırması için `M` (her düğümün bağlantı sayısı) ve `ef_construction` (graf inşa hassasiyeti) gibi parametreleri kullanıcıya ayarlama imkanı da verir [49].

Veri depolama açısından Chroma, tek bir makinede çalışacak şekilde tasarlanmıştır (gömülü veritabanı). Varsayılan kurulumda, veriler **disk üzerinde DuckDB+Parquet** formatında saklanır ve ihtiyaç duyuldukça belleğe yüklenir [50] [51]. DuckDB, analitik sorgular için optimize edilmiş bir SQL veritabanıdır ve Chroma bunu arka planda kullanarak kalıcı depolama sağlar. Küçük ölçekli kullanımlarda ise Chroma tamamen bellek içinde (in-memory) çalışarak verileri RAM'de tutabilir ve maksimum hızda arama yapabilir [52]. Bu ikili mod (kalıcı veya bellek içi) geliştiricilere esneklik sunar; örneğin prototip aşamasında bellek içi kullanılabilir, üretimde büyük veriler için disk moduna geçilebilir. Chroma istemcisi, eklenecek vektörleri ve isteğe bağlı olarak metinsel **metadataları** koleksiyonlar

halinde kaydetmeye imkan tanır; böylece arama sonuçları sadece vektöre değil, ilgili belge kimliği veya metnine ulaşmayı sağlar.

RAG ile Entegrasyonu: Chroma, RAG mimarilerinin doğal bir parçası haline gelmiştir. Birçok açık kaynak RAG uygulaması (ör. LangChain ile yapılan örnekler) **varsayılan vektör deposu** olarak Chroma'yı kullanmaktadır. Bunun sebepleri arasında Chroma'nın kurulum ve kullanım kolaylığı ile açık kaynak oluşu sayılabilir. `pip install chromadb` komutuyla hızlıca kurulabilen bu veritabanı, kod içinde birkaç satırla başlatılıp kullanılabilir durumdadır [53] [50]. Örneğin, LangChain'de `VectorStore` arayüzü altında Chroma sürücüsü mevcuttur; geliştirici, anahtarlarını belirledikten sonra embedding'leri Chroma'ya ekleyip `similarity_search` fonksiyonuyla en benzeyen vektörleri alabilir. Chroma'nın **harici kaynak gerektirmemesi** (cloud servisine ihtiyaç olmaması) de geliştirme sürecinde hız ve maliyet avantajı sunar. Akademik çalışmalarda da Chroma sıkça tercih edilmektedir; 2024'te yayınlanan birkaç RAG araştırmasında, altyapıda ChromaDB kullanıldığı not edilmiştir [54]. Bu, Chroma'nın RAG teknolojisiyle o kadar özdeşleştiğini gösterir ki, RAG ekosisteminde adeta bir *standart araç* mertebesine ulaştığı söylenebilir.

Diğer Vektör Veritabanlarıyla Karşılaştırma: Son 5 yılda vektör veritabanları alanında birçok ürün ortaya çıktı (Pinecone, Weaviate, Milvus, Qdrant, Vespa, vb.). Chroma'yı bunlarla karşılaştırdığımızda birkaç belirgin fark göze çarpıyor:

- *Kurulum ve Mimari:* Chroma tek düğümlü, gömülü bir veritabanıdır (tıpkı SQLite gibi uygulamaya entegre çalışabilir) [55]. Buna karşın diğer pek çok çözüm istemci-sunucu mimarisinde çalışır; yani ayrı bir servis olarak ayağa kaldırmak gerekir. Örneğin, Qdrant veya Weaviate, Docker ile çalıştırılan servisler olup API üzerinden iletişim kurar. Chroma'nın gömülü olması, küçük ve orta ölçekli uygulamalarda basitlik sağlar, ancak **yatay ölçeklenebilirlik** konusunda sınır anlamına gelir – tek makine belleği ve CPU'su ile sınırlıdır.

- *Performans:* Yapılan bağımsız benchmark'lar, Chroma'nın **düşük eş zamanlılık** durumlarında çok hızlı olduğunu ancak yüksek eş zamanlı isteklerde zorlandığını göstermektedir. Sameer Kankute'nin 2024'teki testinde, tek bir sorguda Chroma **en düşük gecikme sürelerini** vermiştir (0.39 saniye ortalama) [56]. Ancak aynı anda 100 sorgu yapıldığında ortalama cevap süresi ~23 saniyeye çıkarak ciddi bir yavaşlama görülmüştür [56] [57]. Buna karşılık, PostgreSQL + PGVector eklentisi kullanan sistemin tek sorguda daha yavaş (~1.9s) olmasına rağmen 100 paralel sorguda ortalamada ~9 saniyede kaldığı ölçülmüştür [58] [59]. Yani **Chroma yüksek yoğunlukta tutarlılığını yitirip yavaşlarken**, PGVector gibi çözümler daha istikrarlı ölçeklenebilmektedir. Hatta DuckDB'nin vektör arama eklentisi testte neredeyse hiç ölçeklenemeyerek 100'lü sorgularda pratik olmaktan çıkmıştır (saniyeler değil dakikalar mertebesine düşmüştür) [60] [61]. Chroma'nın bu sınırlı eş zamanlılık performansının temel nedeni, altında yatan **veritabanı motorunun (varsayılan olarak SQLite)** kilit mekanizmasıdır. SQLite dosya tabanlı olduğundan, aynı anda çoklu okuma/yazma işlemlerinde dosya kilidi nedeniyle beklemeler olur ve işlemler seri hale gelir [62]. PostgreSQL tabanlı PGVector ise her bağlantıyı ayrı işlemci çekirdeğinde yürüterek paralelizmi daha iyi kullanır [63] [64]. Bu nedenle Chroma, **tek kullanıcı veya düşük yük** altında çok hızlı olsa da, **yüksek trafikli uygulamalarda** alternatiflerin gerisinde kalabilir [65].

- *Özellikler:* Vektör veritabanları genelde *sadece vektör ve metadata* saklarken, bazıları anahtar kelime eşleştirme (hibrit arama), coğrafi filtreleme, graf sorguları gibi ek özellikler sunar. Chroma şu an için oldukça minimal bir arayüze sahip: vektör ekleme, benzer vektör arama, metadata filtreleme gibi temel işlevleri yapıyor. Örneğin **Weaviate**, hem vektör benzerlik hem sembolik filtreleri birleştiren *hybrid search* desteğine sahip ve hatta içindeki vektör verilerini skolemleştirme (graf bağlantıları kurma) imkanı veriyor. **Milvus** (ve bulut sürümü Zilliz Cloud)

milyarlarca vektörü bölüp kümeleyecek şekilde ölçeklenebiliyor ve CPU/GPU hızlandırmaları sunuyor. **Pinecone** tamamen yönetilen bir hizmet; ölçekleme detayını kullanıcıdan gizleyip tek bir API ile sınırsız vektör barındırabildiğini iddia ediyor, ancak kapalı kaynak. **Qdrant** açık kaynak olup Chroma gibi HNSW kullansa da Rust ile yazılmış yüksek performanslı bir sunucu olarak tasarlanmış, ayrıca koleksiyonlar arası dağıtık ölçeklemeye başlamış durumda. Chroma ise 2023 itibariyle büyük ölçüde Python ağırlıklı bir kod tabanına sahip, hız kritiklik noktalarında Rust kullanılmıştır [66]. Örneğin HNSW algoritması için HNSWLib kütüphanesi (C++ ile yazılmış) entegre edilmiştir [67]. Ayrıca ilerleyen sürümlerde Chroma ekibi, kalıcı depolamada SQLite yerine **ClickHouse** gibi kolonarlığa sahip ve daha yüksek yazma okuma bant genişliği sunan sistemlere geçişi incelemiştir [67]. Nitekim Chroma'nın şu anki mimarisi Python/TypeScript arayüzünün altında, opsiyonel olarak ClickHouse (C++ tabanlı hızlı bir analitik veritabanı) ve HNSWLib kombinasyonunu kullanabilmektedir [67].

Chroma'nın Artıları ve Eksileri: Özetlemek gerekirse, Chroma **küçük ve orta ölçekli LLM projeleri için biçilmiş kaftandır**: Kolay kurulumu, açık kaynak oluşu, tek makinede çalışabilmesi ve geliştirici dostu arayüzü sayesinde prototipten üretime geçişte hızlı yol aldırır. HNSW tabanlı arama sayesinde milyonlarca vektörde milisaniye-mertebesinde sonuç döndürebilir [68]. Ayrıca istenirse bellek içi modda çalışarak maksimum hız elde edilebilir. DuckDB tabanlı kalıcı depolama, uygulamayı kapatıp açtıktan sonra da verilerin korunmasını sağlar [51]. Bununla birlikte, **Chroma ölçek konusunda temkinli yaklaşılması gereken bir araçtır**: Tek sunucu sınırı, yüksek eşzamanlı sorgularda tıkanma riski, ve henüz olgunlaşmakta olan bir kod tabanı oluşu, kritik uygulamalarda dikkat gerektirir. Halihazırda finansal destek de bulan (2023'te 18 milyon dolar tohum yatırımı aldı [69]) Chroma projesi, gelecekte kurumsal düzeyde yönetilen bir hizmet de sunmayı planlamaktadır [70]. Proje yol haritasında, arama sonuçlarının alaka düzeyini değerlendiren eklentiler, ölçeklenebilirlik geliştirmeleri ve kullanıcı dostu arayüzler bulunmaktadır [70]. Sonuç olarak, Chroma vektör veritabanı, **RAG ekosisteminin pratik ve verimli bir bileşeni** olarak öne çıkmış; ancak kullanım senaryosuna göre diğer veritabanlarıyla karşılaştırılarak seçilmesi gereken bir araçtır. Tekil sorgularda ve prototiplemede **çok hızlı ve kullanışlı**, yüksek yük altında ise **dikkatli konfigürasyon** veya alternatif gerektiren bir çözümdür [65].

5. Semantik Analiz ve Embedding Tabanlı Yöntemler

Semantik analiz, doğal dildeki metinlerin *anlamsal ilişkilerini* ve *içerdiği anlam örüntülerini* inceleyen geniş bir alanı ifade eder. Son 5 yılda, *embedding* tabanlı yöntemlerin yükselişiyle semantik analizde devrim niteliğinde gelişmeler yaşanmıştır. Artık kelime, cümle veya belge düzeyinde metinleri sabit boyutlu vektörlere dönüştürerek makine öğrenimiyle anlam işlemek olağan hale gelmiştir. Bu bölümde, **embedding tabanlı semantik benzerlik**, **örüntü çıkarımı (pattern extraction)**, **vektör uzayı işlemleri** ve bu yaklaşımların güncel NLP uygulamalarındaki rolünü ele alıyoruz.

Embedding Tabanlı Semantik Benzerlik: Metinleri vektör uzayında temsil etmenin en önemli avantajı, **anlamsal benzerliği matematiksel olarak ölçebilme** imkanı sunmasıdır. İki metnin anlamca ne kadar yakın olduğunu, onların embedding vektörleri arasındaki **kozinüs benzerliği** veya **öklid uzaklığı** gibi metriklerle hesaplayabiliriz. Özellikle *kozinüs benzerliği*, vektörlerin açısal yakınlığını ölçerek metin uzunluğundan bağımsız bir benzerlik skoru verir ve NLP'de yaygın olarak kullanılır [71] [72]. Bu sayede, örneğin "araba" ve "otomobil" kelimelerinin embedding'leri yüksek bir kozünüs benzerliği (yakın açılar) gösterecek, "araba" ile "uçak" arasında daha düşük benzerlik çıkacaktır. Embedding tabanlı benzerlik, klasik anahtar kelime eşleşmesine göre çok daha esnektir: Eşanlamlılar, farklı biçimlerde ifade edilen benzer kavramlar yüksek skorlarken anlamsal açıdan ilgisiz metinler düşük skor alır. Bu yaklaşım, **metin çiftlerinin benzerlik sıralaması** (Semantic Textual Similarity - STS) görevlerinde insan yargısına yakın sonuçlar üretmekte ve son yıllarda birincil yöntem haline gelmiştir [73]. Örneğin, bir STS benchmark'ında embedding tabanlı sistemlerin çıktı benzerlik skorları ile insanlarca verilen benzerlik puanları arasında yüksek korelasyon gözlenmiştir [74].

Bu teknik pratikte **plagiarizm tespiti, metin kümeleme, öneri sistemleri** gibi pek çok alanda kullanılır. Bir belge kümesinde birbirine çok benzeyenleri tespit etmek (ör. mükerrer cümleleri bulma) embedding benzerlikleriyle yapılabilir. Tavsiye sistemlerinde, bir kullanıcının okuduğu makaleye en benzer diğer makaleleri bulmak için embedding'ler kullanılabilir. Hatta siber güvenlikte, normal kullanıcı davranışlarının embedding'leri ile anormal davranışların embedding'leri kıyaslanarak anomali tespiti yapılmaktadır – **embedding'lerin benzerlik hesaplaması hem benzerlik hem farklılıkları vurgulayabilir** [75] [76] . Örneğin, normal trafik modellerine benzemeyen (dissimilar) ağ etkinlikleri varsa bunlar olası saldırı olarak bayraklanabilir [77] . Bu, embedding tabanlı benzerlik ölçümünün sadece benzer içeriği değil, aynı zamanda **alışılmadık/sıradışı içeriği** de saptamaya yarayabileceğini gösterir.

Örüntü Çıkarımı ve Vektör Uzayında İşlemler: Embedding'lerin sayıltısal niteliği, dilsel örüntüleri keşfetmeyi de mümkün kılar. *Vektör uzayı işlemleri* denildiğinde en bilinen örnek, **analojiler**dir: Klasik örnekte "kral – adam + kadın = kraliçe" işlemi, Word2Vec vektörlerinde başarılı şekilde işe yaramış ve büyük ilgi görmüştü. Bu, vektörlerin belli boyutlarında toplumsal cinsiyet gibi yönler tutulabildiğini ima ediyordu. Türkçe'de de benzer şekilde "ankara – türkiye + fransa ≈ paris" gibi şehir-başkent ilişkileri vektör aritmetiğiyle ortaya çıkarılabilir. Bu fenomen her zaman kusursuz olmasa da, embedding'lerin çok boyutlu uzayında anlamsal yönlerin yakalanabildiğini gösteren çarpıcı bir örüntü çıkarımı yöntemidir. Yine vektör uzayında **kümeleme (clustering)** algoritmaları uygulandığında, genellikle benzer anlamdaki metinler aynı gruplarda toplanır. Örneğin, haber makalelerinin embedding'lerini k-means ile kümelerseniz spor haberleri, ekonomi haberleri, magazin haberleri ayrı kümelerde toplanabilir – model bunların konularını "etiketlenmese bile" salt istatistiksel benzerlikle gruplayabilir. Bu yöntem, **konu tespiti, özetleme için önemli cümleleri bulma** gibi işlerde örüntü keşfi sunar. Nitekim bir araştırma, *cümle embedding'lerine dayalı bir yöntemle* kısa metinleri kümelendirip, mevcut yöntemlerin yetersiz kaldığı durumlarda bile anlamlı gruplar elde edilebildiğini ortaya koymuştur [78] .

Bağlamsal örüntüler: BERT gibi modellerin embedding'leri daha da zengindir; cümle içindeki ilişkileri yansıtır. Örneğin, bir cümlede özne-nesne ilişkisini vurgulayan bir bileşen, farklı cümlelerde benzer ilişkiler görüldüğünde yakın vektörler üretebilir. Bu sayede embedding'ler örneğin *dilbilgisel roller* veya *duygu tonları* gibi örüntüleri bile ayırt edebilir. Güncel çalışmalarda, büyük dil modellerinin iç temsil uzayında belirli nöronların veya boyutların *belirli anlamsal işlevlerle korele* olduğu keşfedilmektedir. Bu da, vektör uzayı üzerinde kontrollü işlemler yaparak (ör. belirli bir yönü vektörden çıkararak) model çıktısının değiştirilebileceğini, yani istenmeyen bir örüntünün bastırılabileceğini göstermektedir. Örneğin, metin embedding'lerinde "negatif duygu" yönünü tespit edip çıkarırsanız, vektörün temsil ettiği cümlenin duygusunu nötrleştirebilirsiniz – bu da *stil transferi* gibi alanlarda kullanılmaktadır.

Güncel NLP Uygulamaları: Semantik analiz, embedding'ler sayesinde pek çok pratik uygulamada iskelet teknoloji haline geldi:

- **Semantik Arama:** Arama motorları artık anahtar kelime eşleşmesinin ötesine geçiyor. *Embedding tabanlı semantik arama*, kullanıcının sorgusunu ve tüm indekslenmiş dokümanları vektör uzayına alıp, en yakın vektörlü dokümanları getirerek çalışıyor [79] [80] . Bu sayede kullanıcı tam kelimeyi yazmasa da benzer anlamdakiler bulunuyor. Örneğin "kredi kartı ödemesi gecikti" diye aratan birine, içinde "ödeme gecikmesi" geçen bir metin veya "kredi kartı borcu gecikme faizi" konulu bir makale getirilebilir – çünkü embedding'ler bu kavramların yakınlığını yakalar. Semantik arama, özellikle kurumsal bilgi içeriğinde (dokümanasyon, sıkça sorulan sorular) ve e-ticarette ürün aramalarında öne çıkıyor. Ayrıca RAG sistemlerinde de arka planda genellikle embedding tabanlı arama vardır (LLM'e verilecek pasajları bu yöntemle seçiyoruz). Hibrit yaklaşımlar (anahtar kelime + embedding bir arada) en yüksek başarımları sağlıyor [81] .

- **Metin Sınıflandırma:** Geleneksel yöntemler metinleri sıklık özellikleriyle temsil ederken, artık genelde ön-eğitimli bir modelden alınan embedding'ler özellik olarak kullanılmakta. Örneğin bir

duygu analizi modelinde, BERT'in [CLS] token embedding'i alınarak bunun üzerinde bir lojistik regresyon eğitilebiliyor. Bu yaklaşım, ham kelime sayma temelli özelliklere göre çok daha yüksek doğruluk getiriyor, çünkü embedding'ler kelimenin bağlamını ve nüansını içeriyor. Dahası, **sıfır-atış (zero-shot) sınıflandırma** da embedding'lerle mümkün hale geldi: Örneğin *Sentence-BERT* modeliyle, etiket açıklamalarının embedding'lerini ve cümlenin embedding'ini karşılaştırıp en yüksek kozünüs benzerliğine göre etiketi seçmek gibi yöntemler, hiç o dilde veri görmeden bile makul sınıflandırma yapabiliyor. Son yıllarda yayınlanan çalışmalar, bilimsel makalelerin konu sınıflandırmasından müşteri geri bildirimlerinin kategorizasyonuna kadar birçok alanda embedding tabanlı sınıflandırma yaklaşımlarının başarılı olduğunu gösteriyor.

- **Anlamsal Özetleme ve Bilgi Çıkarma:** Embedding'ler, uzun bir metindeki en önemli cümleleri seçmede de kullanılıyor. *Extractive summarization* denilen yöntemde, tüm cümleler embedding'e dönüştürülüp aralarındaki benzerlik matrisine bakarak merkezi konumdakiler bulunabiliyor. Yine, anahtar kelime/anahtar cümle çıkarma (keyphrase extraction) problemlerinde, embedding tabanlı derin öğrenme yöntemleri klasik istatistiksel yöntemleri geride bıraktı [82]. Örneğin, 2024'te yayımlanan bir derleme, bağlamsal embedding kullanan derin model tabanlı anahtar kelime çıkarıcıların, bağlamı tam hesaba katamayan eski yöntemlere kıyasla çok daha yüksek F1 skorlarına ulaştığını vurgulamıştır [82]. Bu da, metin içinde insanın sezgisel olarak *"önemli"* gördüğü kavramları, modelin de embedding'ler yardımıyla yakalayabildiğini gösteriyor.

- **Çapraz Modallar ve Diğer Uygulamalar:** Semantik analiz sadece metinle sınırlı değil; resimlerin, seslerin veya videoların da embedding'leri çıkarılıp bunlar üzerinden benzerlik veya örüntü analizi yapılabiliyor. Örneğin, bir görsel arama motoru resmi *CLIP modelinin* vektör uzayına alıp, metin sorgusunu da aynı uzaya alarak en yakın görselleri buluyor. Bu, görsel ve dil uzaylarını ortak bir semantik alana yerleştirerek mümkün oluyor. Benzer şekilde, **sohbet robotları** bir kullanıcının sorusunu embedding yapıp bilgi tabanındaki soru-cevap çiftleriyle karşılaştırarak en uygun cevabı getirmeyi hızla yapabiliyor. **Çeviri** alanında, farklı dil cümlelerini aynı manaya geldiklerinde yakına düşürecek ortak embedding uzayları (örn. LASER, LaBSE modelleri) sayesinde çeviri eşleşmeleri bulunabiliyor.

Son Gelişmeler: 2020'lerin ortasına gelirken, semantik analiz yöntemleri giderek daha fazla *büyük dil modelleriyle iç içe* geçiyor. Artık LLM'ler bir metnin derin semantik analizini kendileri üretken bir şekilde yapabildiğinden (ör. bir metinde anlatılmak isteneni insan gibi özetleyip çıkarabiliyorlar), embedding tabanlı *hesaplamalı* yöntemler ile *üretken LLM* yöntemleri birlikte kullanılıyor. Örneğin bir belge kümesinde önce embedding'lerle kümeler bulunup sonra her küme için bir LLM'den açıklama ürettirmek gibi birleşik yaklaşımlar görülüyor. Ayrıca *"bellek"* kavramı, embedding'ler sayesinde LLM'lere entegre edildiğinden, uzun konuşmaların özünü anlamak veya kişisel örüntüleri yakalamak mümkün oluyor. Tüm bunlar, semantik analizin artık **daha dinamik ve bağlama duyarlı** bir hale geldiğini gösteriyor.

Özetle, embedding tabanlı semantik analiz, **doğal dil anlamını sayısal uzayda işleyen** güçlü bir araç seti sunmaktadır. Anlamsal benzerlik ölçümü, örüntü keşfi ve vektör işlemleri sayesinde, metinler arasındaki ilişkiler daha önce mümkün olmayan biçimde ortaya çıkarılabilmiştir. Bu yaklaşımlar güncel NLP uygulamalarının kalbinde yer almakta; arama motorlarından sohbet robotlarına, belge özetlemeden sahte haber tespitine kadar pek çok sistemde **anlam tabanlı hesaplamalar** embedding'ler aracılığıyla gerçekleştirilmektedir. Geleneksel yöntemlerin ötesinde, derin öğrenme temelli gömme teknikleri bağlam bilgisini ve nüansları yakalayarak semantik analizde insan düzeyine yaklaşan performanslar ortaya koymuştur [83]. Bu eğilimin önümüzdeki yıllarda da sürmesi ve daha bütüncül, çok modlu semantik analiz sistemlerinin geliştirilmesi beklenmektedir.

Kaynaklar:

1. RAG (Retrieval-Augmented Generation) kavramı ve kullanım alanları – Nvidia Blog makalesi [1] [9] [84] [15]; IBM Research açıklayıcı yazısı [85] [7] [4] [3]; RAG literatür taraması (Gao ve ark., 2024) [11] [10].
2. Türkçe word embedding çalışmaları – 2024 kapsamlı analizi (ArXiv) [23] [27]; önceki araştırmalar özeti [20] [22] [25]; açık kaynak Türkçe embedding depoları [23].
3. LangChain framework – Medium makalesi (L. Kilpatrick, 2023) [36] [38] [35]; LangChain resmi sitesi istatistikleri [40]; LangChain belgelendirmesi ve blog yazıları.
4. Chroma vektör veritabanı – Medium teknik tanıtım (N. Kukreti, 2025) [46] [51] [68]; Newtuple benchmark yazısı [57] [62] [65]; Chroma Wikipedia bilgileri [44]; SiliconANGLE haber yazısı (2023) [13] [52].
5. Semantik analiz ve embedding uygulamaları – PyImageSearch semantik arama anlatımı [71] [72]; Chroma kullanımı örnekleri (embedding ile benzerlik) [75] [77]; Wiley derleme makalesi (2024) [82]; çeşitli araştırma makaleleri ve blog içerikleri (Medium, 2023) [81] [80].

---

[1] [9] [15] [16] [17] [84] What Is Retrieval-Augmented Generation aka RAG | NVIDIA Blogs
https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/

[2] [3] [4] [5] [6] [7] [8] [85] What is retrieval-augmented generation (RAG)? - IBM Research
https://research.ibm.com/blog/retrieval-augmented-generation-RAG

[10] [11] [12] [18] [2312.10997] Retrieval-Augmented Generation for Large Language Models: A Survey
https://ar5iv.labs.arxiv.org/html/2312.10997v5

[13] [14] [52] [70] [75] [76] [77] Chroma funding: Database provider raises $18M for AI-Powered Database - SiliconANGLE
https://siliconangle.com/2023/04/06/chroma-bags-18m-speed-ai-models-embedding-database/

[19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] A Comprehensive Analysis of Static Word Embeddings for Turkish
https://arxiv.org/html/2405.07778v1

[31] Introduction | LangChain
https://python.langchain.com/docs/introduction/

[32] [33] [34] [35] [36] [37] [38] [39] [41] [43] What is Langchain and why should I care as a developer? | by Logan Kilpatrick | Around the Prompt | Medium
https://medium.com/around-the-prompt/what-is-langchain-and-why-should-i-care-as-a-developer-b2d952c42b28

[40] [42] LangChain
https://www.langchain.com/

[44] [45] [54] [66] [69] Chroma (vector database) - Wikipedia
https://en.wikipedia.org/wiki/Chroma_(vector_database)

[46] [47] [48] [49] [50] [51] [53] [68] ChromaDB. ChromaDB is an open-source vector... | by Nishtha kukreti | Medium
https://medium.com/@nishthakukreti.01/chromadb-fb20279e244c

[55] [67] Vector databases (1): What makes each one different? • The Data Quarry
https://thedataquarry.com/blog/vector-db-1/

[56] [57] [58] [59] [60] [61] [62] [63] [64] [65] Vector DB Benchmarks: ChromaDB vs pgVector vs DuckDB
https://www.newtuple.com/post/speed-and-scalability-in-vector-search

71 72 79 Implementing Semantic Search: Jaccard Similarity and Vector Space Models - PyImageSearch
https://pyimagesearch.com/2024/07/22/implementing-semantic-search-jaccard-similarity-and-vector-space-models/

73 74 Enhancing Negation Awareness in Universal Text Embeddings - arXiv
https://arxiv.org/html/2504.00584v1

78 Experimental study on short-text clustering using transformer-based ...
https://pmc.ncbi.nlm.nih.gov/articles/PMC11157522/

80 81 Unlocking the Potential of Search Engines with LLMs and Embeddings
https://ai.plainenglish.io/unlocking-the-potential-of-search-engines-with-llms-and-embeddings-9f0ca480ac6d

82 83 Deep learning and embeddings-based approaches for keyphrase extraction: a literature review | Knowledge and Information Systems
https://link.springer.com/article/10.1007/s10115-024-02164-w