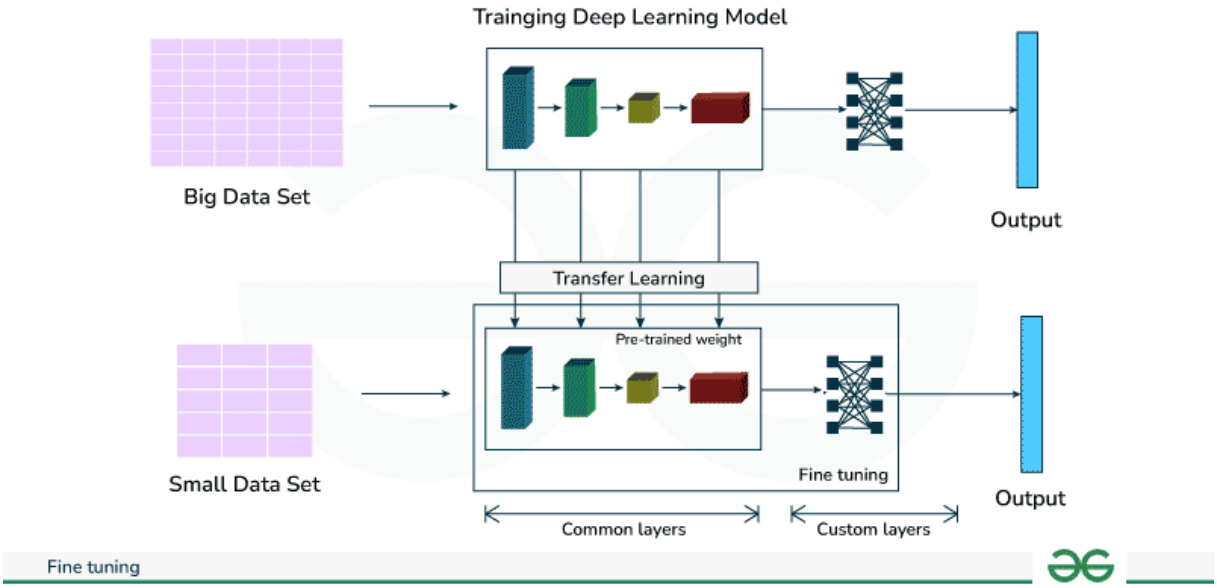
Fine-tuning Nedir? Neden Gereklidir?

Fine-tuning (ince ayar), önceden eğitilmiş bir dil modelini belirli bir görevi, alanına özgü girdiler üzerinde daha yüksek doğrulukla gerçekleştirecek şekilde uyarlama sürecidir. GPT gibi genel amaçlı modellerin, tıbbi özetleme veya müşteri hizmetleri otomasyonu gibi alanlarda son derece etkili hale gelmesini sağlar.

Bu yaklaşım, mevcut bir modelin yeni fakat ilişkili bir görevi yerine getirecek şekilde güncellenmesini ifade eden bir **transfer öğrenme (transfer learning)** türüdür. Transfer öğrenme, bir modeli sıfırdan yeniden eğitme ihtiyacını azaltmak için önceki eğitimin üzerine inşa edilir. Fine-tuning ise bu yaklaşımı geliştirerek, daha küçük miktarda göreve özgü veriyle modeli uyarlamayı ve performansını artırmayı sağlar.



Model, kısa bir eğitim aşaması boyunca yeni örneklerle beslenir. Bu süreçte model, tahminleri etkileyen iç parametrelerini — yani ağırlıklarını — genel dil yeteneklerini kaybetmeden ayarlar. Sonuç olarak, özetleme veya bilgi erişimi gibi yüksek etkili iş görevlerine uyarlanmış bir model elde edilir.

Fine-tuning nasıl çalışır?

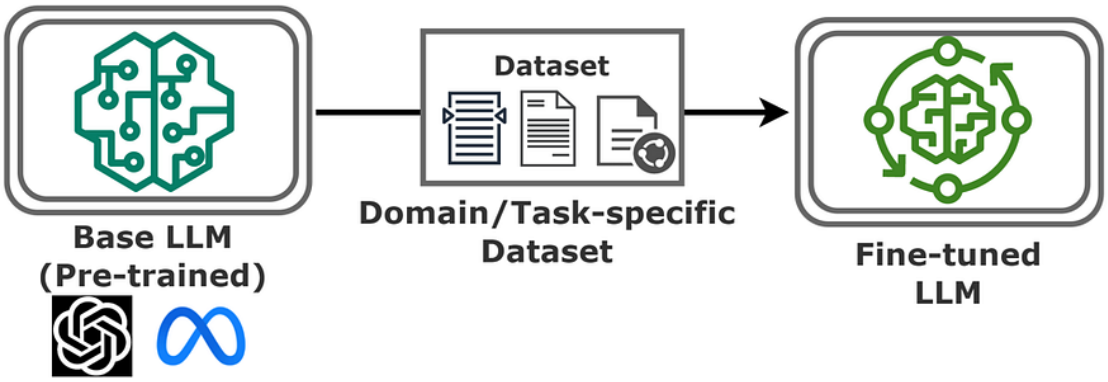
Fine-tuning sürecine başlamak karmaşık olabilir; bu nedenle sürecin tipik adımlarına genel bir bakışla başlamak faydalıdır.

Aşağıda, bir modeli belirli görevler için ince ayar yaparken izlenen tipik adımlar yer almaktadır:

- **Önceden eğitilmiş bir model seçmek:**

Fine-tuning, büyük ölçekli veri kümeleri üzerinde önceden eğitilmiş bir modelle başlar. Bu tür modeller Hugging Face gibi platformlarda bulunur ve metin veya görsel gibi veri türlerine göre gruplandırılır.

Model seçimi; modelin mimarisine, hangi veriyle eğitildiğine ve kabul ettiği giriş formatına bağlıdır. Ekipler, göreve ve teknik altyapıya en uygun modeli değerlendirir ve açık kaynaklı (open-source) ya da ticari kapalı kaynaklı (closed-source) LLM'ler arasında seçim yapar.



- **Veriyi hazırlamak ve formatlamak:**

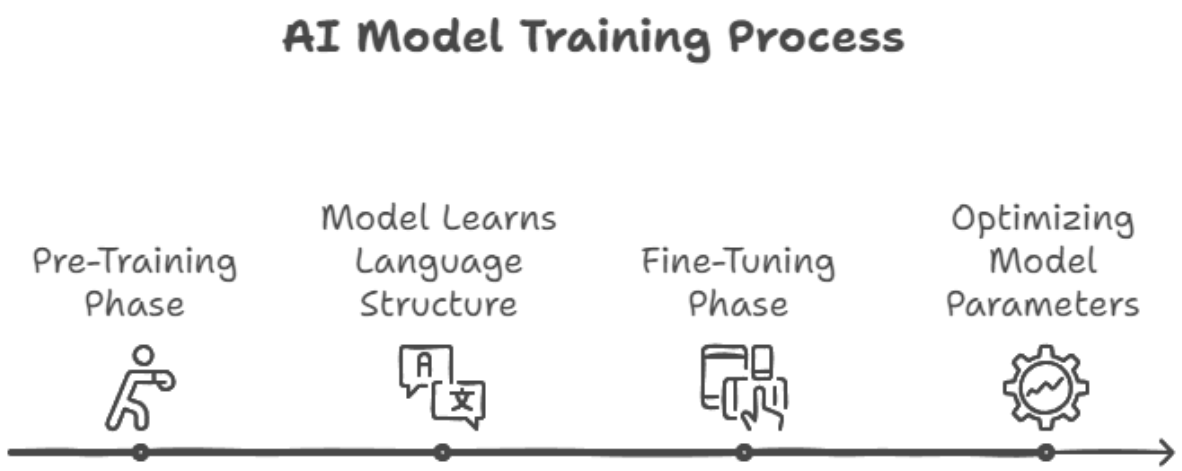
Model, öğrenmesi beklenen görevi gösteren etiketli örneklerle beslenir. Veri temizlenir; tutarsızlıklar, tekrarlar ve biçim hataları giderilir. Ardından veri, eğitim ve doğrulama (validation) kümelerine ayrılır.

Eğitim örnekleri, modelin gerektirdiği giriş-çıkış yapısına uygun biçimde formatlanmalıdır. Model kartlarında genellikle gerekli talimat formatı belirtilir. JSONL yaygın bir formattır ve her satır bir eğitim örneğini temsil eder.

- **Fine-tuning parametrelerini ayarlamak:**

Veri bilimciler, modelin eğitim verisinden nasıl öğreneceğini belirler.

- Learning rate (öğrenme oranı), modelin her örnekten sonra anlayışını ne kadar güncelleyeceğini kontrol eder.
- Batch size, aynı anda kaç örneğin işlendiğini belirler ve hız ile öğrenme kalitesini etkiler.
- Epoch sayısı, modelin tüm veri kümesini kaç kez baştan sona işleyeceğini ifade eder.



Bu parametrelerin dikkatli dengelenmesi gerekir. Çok düşük ayarlanırlarsa model yeterince öğrenemeyebilir (**underfitting**). Çok yüksek ayarlanırlarsa model eğitim verisini aşırı ezberleyebilir ve yeni veride zayıf performans gösterebilir (**overfitting**).

Çoğu zaman, erken katmanlar dondurulur (freeze edilir) — yani eğitim sırasında güncellenmez — böylece genel bilgi korunur; daha sonraki katmanlar ise göreve özgü olarak ince ayar yapılır.

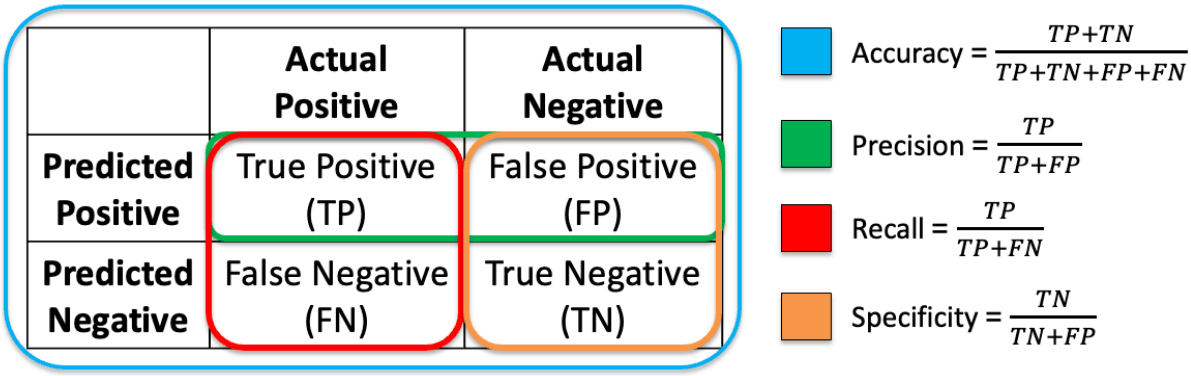
- **Fine-tuning sürecini çalıştırmak**

Kurulum tamamlandıktan sonra ince ayar süreci başlar. Bu işlem genellikle büyük ölçekli işlemleri kaldırabilecek bir bulut platformunda gerçekleştirilir. Model, eğitim verisini batch'ler halinde işler ve verilen örneklere dayanarak iç parametrelerini kademeli olarak ayarlar.

Platform genellikle kaç eğitim adımının tamamlandığı gibi ilerleme bilgileri sunar. Süreç tamamlandığında, modelin yeni bir versiyonu kendine özgü bir kimlikle kaydedilir. Model genel dil yeteneklerini korur ancak artık belirli görevi daha yüksek doğrulukla yerine getirecek şekilde uyarlanmıştır.

- **Model performansını değerlendirmek**

Eğitimden sonra model, doğrulama veri kümesi kullanılarak değerlendirilir. Amaç, modelin daha önce görmediği yeni örneklerde nasıl performans gösterdiğini ölçmektir.



- ✓ **Accuracy (doğruluk)**, modelin ne sıklıkla doğru yanıt verdiğini ölçer.
- ✓ **Precision (kesinlik)**, modelin pozitif tahminlerinin ne kadarının doğru olduğunu gösterir.
- ✓ **Recall (duyarlılık)**, doğru yanıtların ne kadarını başarıyla bulduğunu ifade eder.
- ✓ **F1 skoru**, precision ve recall'u birleştirerek dengeli bir performans ölçütü sunar.

Sonuçları iyileştirmek için ekipler genellikle daha temiz veri veya ayarlanmış parametrelerle fine-tuning sürecini tekrarlar.

- **Fine-tuned modeli yayına almak (deploy etmek)**

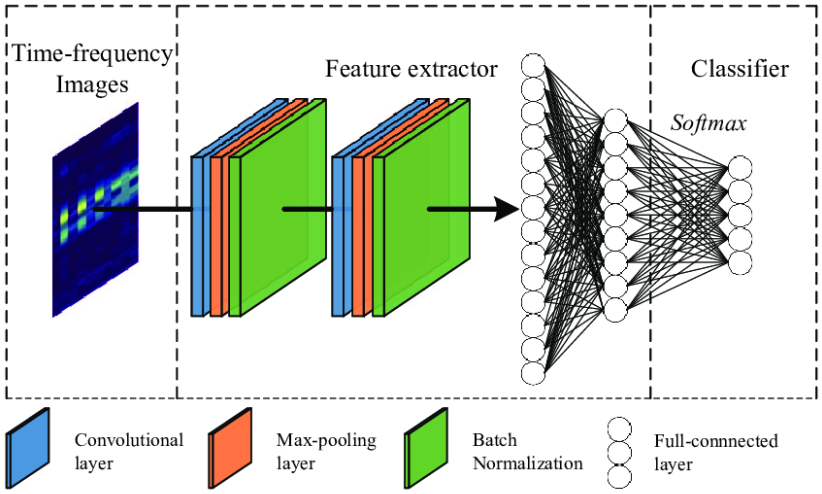
Model değerlendirme aşamasında tatmin edici sonuçlar elde ettiğinde, gerçek zamanlı uygulamalar tarafından erişilebilecek bir sunucuya veya bulut ortamına dağıtılır. Model, fine-tuning sürecinde kazandığı bilgiyi kullanarak yeni girdiler geldiğinde tahmin üretir veya çıktı oluşturur.

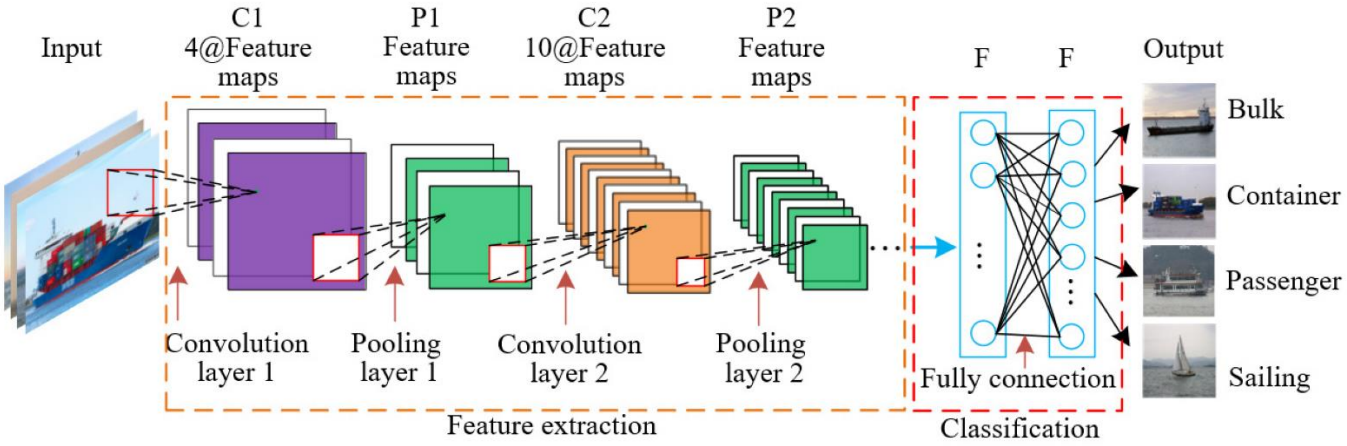
Yayına alındıktan sonra ekipler modelin gerçek dünya koşullarındaki performansını izlemeye devam eder. Zamanla kullanıcı davranışları, işlenen veri türleri veya modelden beklenen görevler değişebilir. Bu değişimler modelin doğruluğunu etkileyebilir. Eğer performans zamanla düşerse, model güncellenmiş örneklerle yeniden fine-tuning yapılarak sistem gereksinimlerini karşılaması sağlanır.

Full fine-tuning vs. Feature Extraction Farkı Nedir?

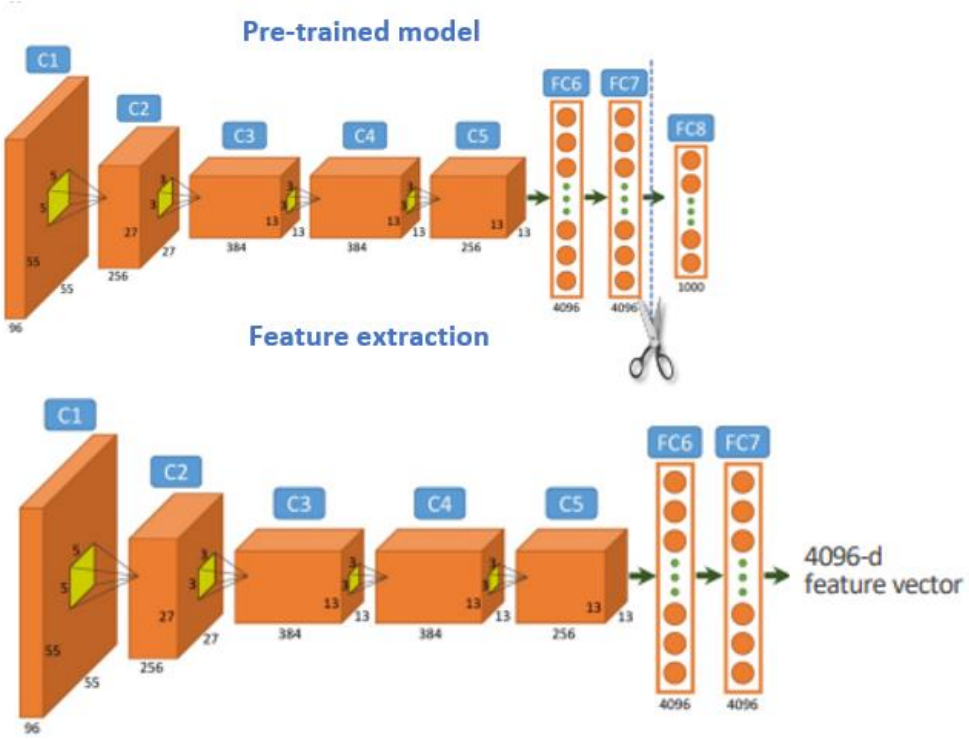
Transfer öğrenme, modern derin öğrenme uygulamalarının temel yaklaşımlarından biridir. Önceden büyük veri kümeleri üzerinde eğitilmiş modellerin yeni görevlere uyarlanması sürecinde iki temel strateji öne çıkmaktadır: **feature extraction (özellik çıkarımı) ve full fine-tuning (tam ince ayar).** Ancak bu iki yöntem arasındaki tercih, yalnızca teknik bir seçim değil; veri setinin büyüklüğü, hedef görev ile kaynak görev arasındaki benzerlik ve mevcut hesaplama kaynakları gibi faktörlere bağlı stratejik bir karardır.

Feature extraction yaklaşımında, önceden eğitilmiş model sabit bir özellik çıkarıcı olarak kullanılır. Modelin erken ve orta katmanlarında öğrenilmiş temsil gücü korunur ve bu katmanların ağırlıkları güncellenmez. Genellikle yalnızca son sınıflandırma katmanı değiştirilir ve yeni göreve uygun bir başlık (classifier head) eklenir.

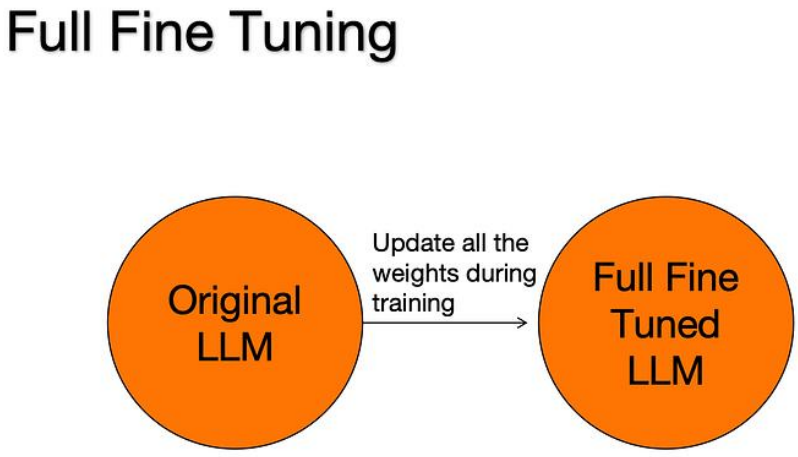




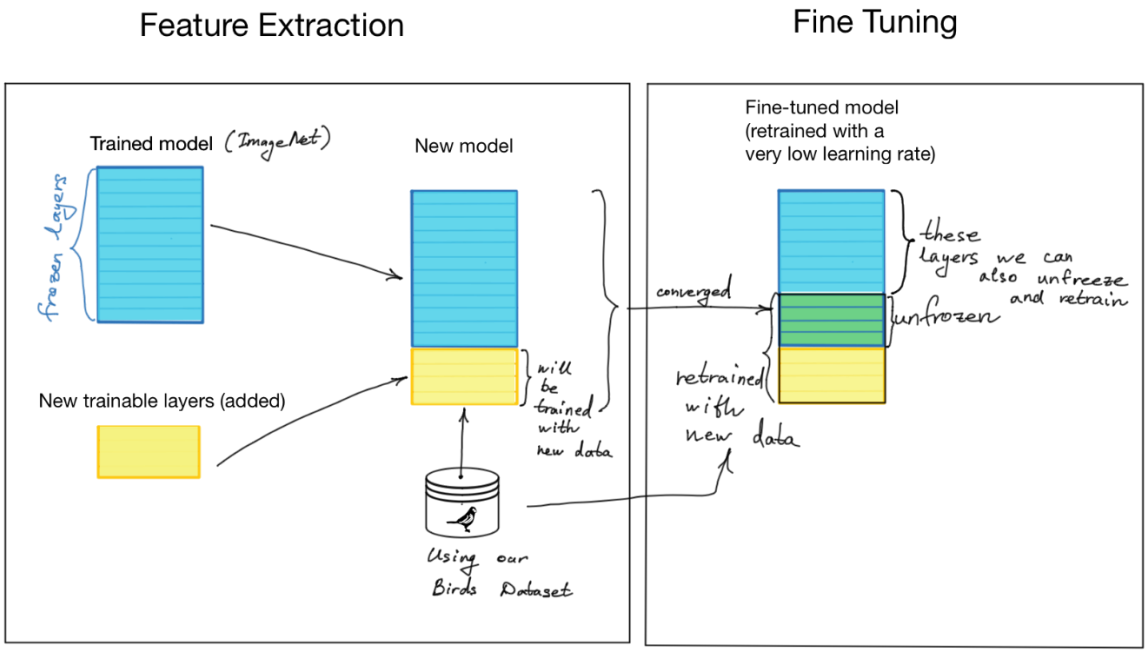
Eğitim sürecinde yalnızca bu yeni katman eğitilir. Bu yöntem hesaplama maliyeti açısından avantajlıdır ve özellikle hedef veri setinin küçük olduğu veya kaynak veriyle yüksek benzerlik taşıdığı durumlarda etkili sonuçlar verir. Bununla birlikte, modelin temsil kapasitesi büyük ölçüde sabit kaldığı için, hedef görev kaynak görevden önemli ölçüde farklıysa performans sınırlı kalabilir.



Full fine-tuning yaklaşımında ise modelin tüm katmanları eğitim sürecine dahil edilir. *Önceden eğitilmiş ağırlıklar başlangıç noktası olarak kullanılır;* ancak eğitim sırasında ağın tüm parametreleri güncellenir.



Bu yöntem, modelin öğrendiği özellikleri hedef görevin gereksinimlerine daha derinlemesine uyarlamasına olanak tanır. Özellikle hedef veri setinin büyük olduğu ve görev yapısının kaynak görevden belirgin şekilde farklılaştığı durumlarda daha yüksek performans elde edilebilir. Ancak bu yaklaşım yüksek hesaplama maliyeti gerektirir ve öğrenme oranının dikkatli ayarlanmasını zorunlu kılar. Aksi halde model, ön eğitim sırasında kazanılmış genel bilgiyi kaybedebilir (catastrophic forgetting).

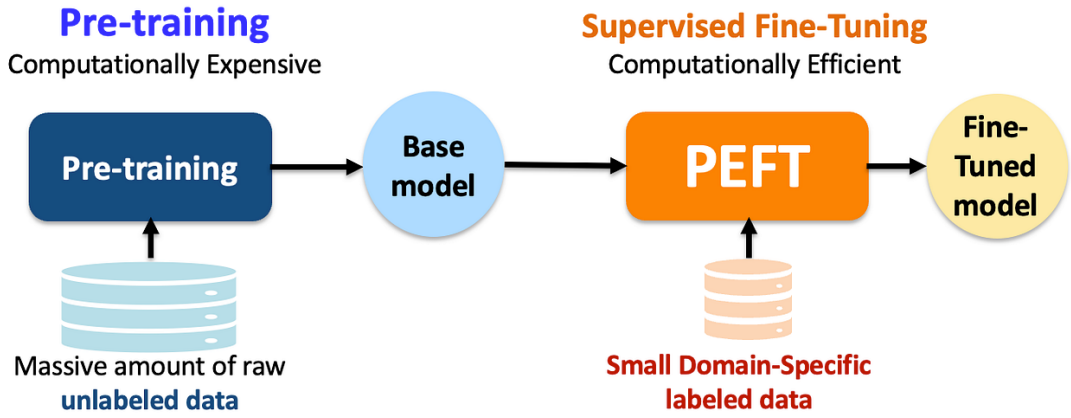


Katmanların öğrenme karakteristiği bu farkın anlaşılmasında kritik rol oynar. Derin sinir ağlarının erken katmanları genellikle genel örüntüleri (kenarlar, temel yapılar, dilde temel sözdizimsel kalıplar) öğrenirken, daha derin katmanlar göreve özgü daha soyut temsiller üretir**. Feature extraction yöntemi bu genel temsilleri sabit tutarken, full fine-tuning yöntemi hem genel hem de görev-özgü temsillerin yeniden şekillendirilmesine izin verir.**

Sonuç olarak, feature extraction daha düşük maliyetli ve hızlı bir uyarlama stratejisi sunarken, full fine-tuning daha yüksek esneklik ve potansiyel performans artışı sağlar. En uygun yaklaşım; veri setinin büyüklüğü, görev benzerliği, overfitting riski ve donanım kapasitesi dikkate alınarak belirlenmelidir. Bu nedenle model uyarlama süreci çoğu zaman deneysel ve yinelemeli bir değerlendirme gerektirir.

PEFT (Parameter-Efficient Fine-Tuning) Nedir?

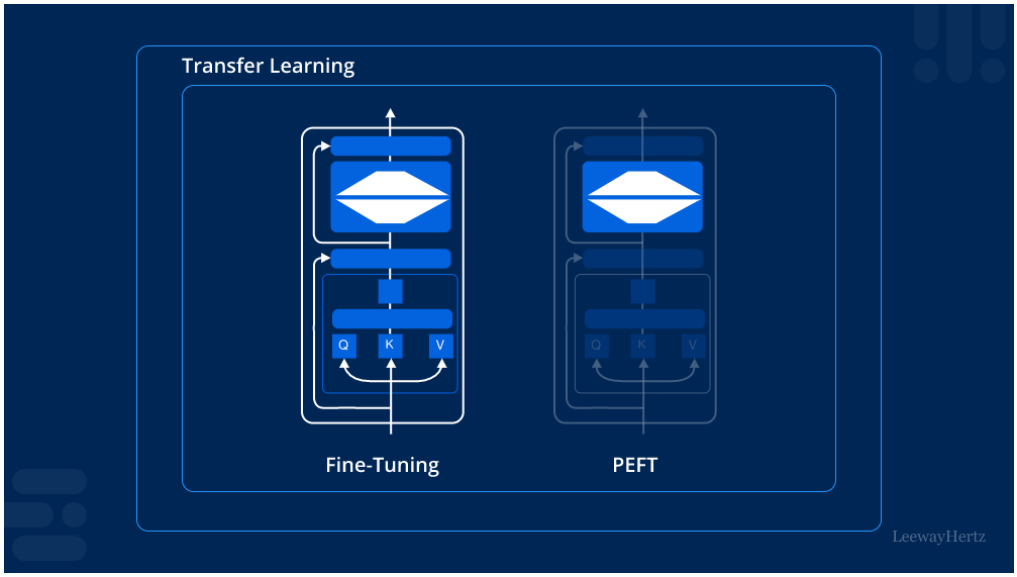
Büyük dil modellerinde tüm parametreleri güncellemek çoğu zaman pratik değildir. Bu noktada Parameter-Efficient Fine-Tuning (PEFT) yöntemleri devreye girer. PEFT, modelin tamamını güncellemek yerine yalnızca küçük bir parametre alt kümesini eğitmeyi amaçlar.



Bu yaklaşım sayesinde:

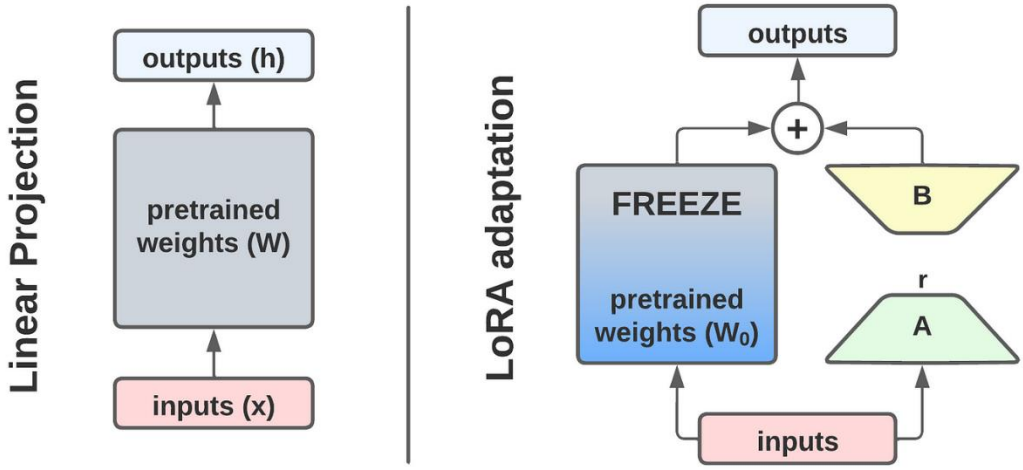
- Eğitim maliyeti azalır
- Bellek kullanımı düşer
- Daha hızlı model güncellemesi sağlanır
- Aynı temel model farklı görevler için küçük ek modüllerle kullanılabilir

PEFT yöntemleri, modern LLM araştırmalarında ölçeklenebilir ve sürdürülebilir model adaptasyonu için kritik öneme sahiptir.

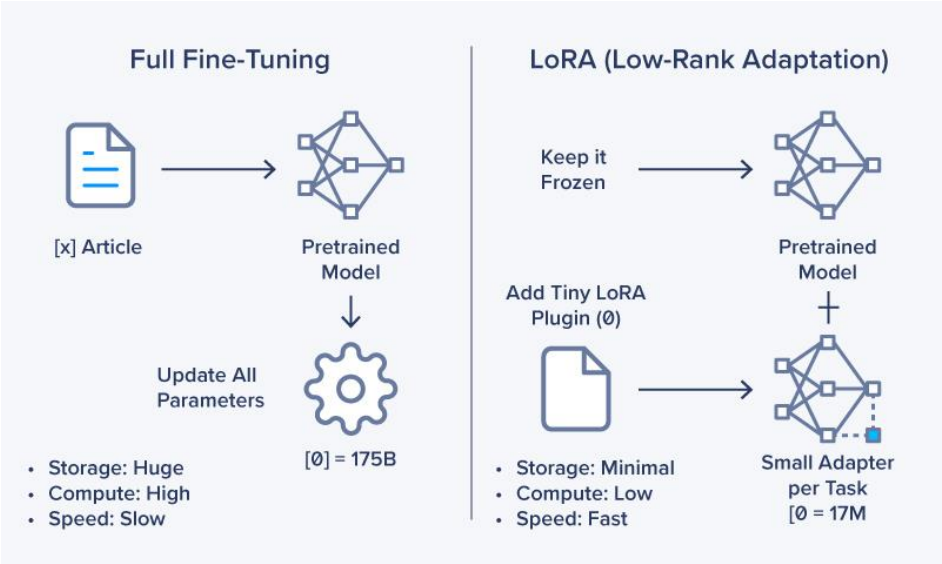


LoRA (Low-Rank Adaptation)

LoRA, PEFT yöntemleri arasında en yaygın kullanılan tekniklerden biridir. Bu yöntemde modelin mevcut ağırlıkları doğrudan değiştirilmez. Bunun yerine, ağırlık güncellemesi düşük dereceli (low-rank) iki matrisin çarpımı şeklinde modellenir.



Matematiksel olarak, büyük bir ağırlık matrisi yerine düşük boyutlu iki küçük matris öğrenilir ve bu matrislerin çarpımı ana ağırlığa eklenir. Böylece öğrenilecek parametre sayısı ciddi biçimde azalır.



LoRA'nın avantajları şunlardır:

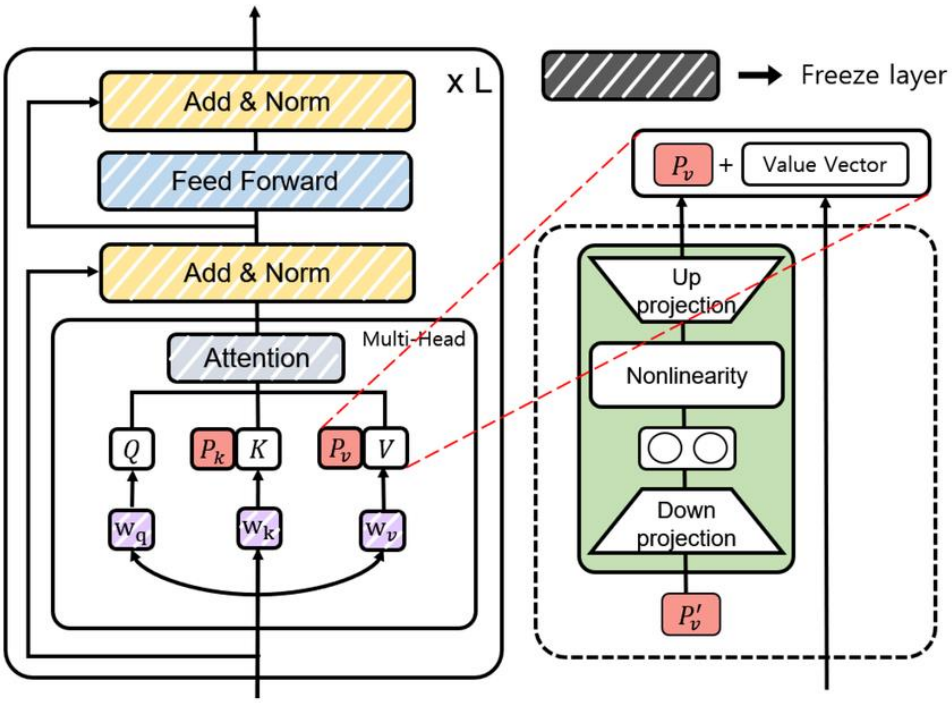
- Çok daha az parametre güncellenir
- GPU bellek ihtiyacı düşer
- Farklı görevler için küçük LoRA dosyaları saklanabilir
- Ana model ağırlıkları sabit kalır

Özellikle attention katmanlarında uygulandığında oldukça verimli sonuçlar verir.

Prefix Tuning

Prefix Tuning yönteminde modelin attention mekanizmasına öğrenilebilir “prefix” vektörleri eklenir. Bu vektörler girişe ek bağlam (context) gibi davranır ve modelin davranışını yönlendirir.

```python
def transformer_block_with_prefix(x, soft_prompt):
    soft_prompt = FullyConnectedLayers(soft_prompt)  # Prefix
    x = concatenate([soft_prompt, x],                # Prefix
                    dim=seq_len)                     # Prefix
    residual = x
    x = self_attention(x)
    x = LayerNorm(x + residual)
    residual = x
    x = FullyConnectedLayers(x)
    x = LayerNorm(x + residual)
    return x
```



Bu yaklaşımda:

- Ana model ağırlıkları değiştirilmez
- Sadece prefix parametreleri güncellenir
- Görev özelinde hafif bir adaptasyon sağlanır

Bu yöntem özellikle metin üretim görevlerinde etkili sonuçlar vermektedir.

Adapter Yöntemi:

Adapter yaklaşımında transformer katmanlarının arasına küçük, daraltılmış ve yeniden genişletilmiş (bottleneck yapıda) modüller eklenir. Eğitim sırasında yalnızca bu adapter katmanları güncellenir.

Avantajları:

- Ana model sabit kalır
- Her görev için ayrı adapter modülü eklenebilir
- Modüler yapı sayesinde esnek kullanım sağlanır

Bu yöntem, çoklu görev (multi-task) senaryolarında oldukça kullanışlıdır.

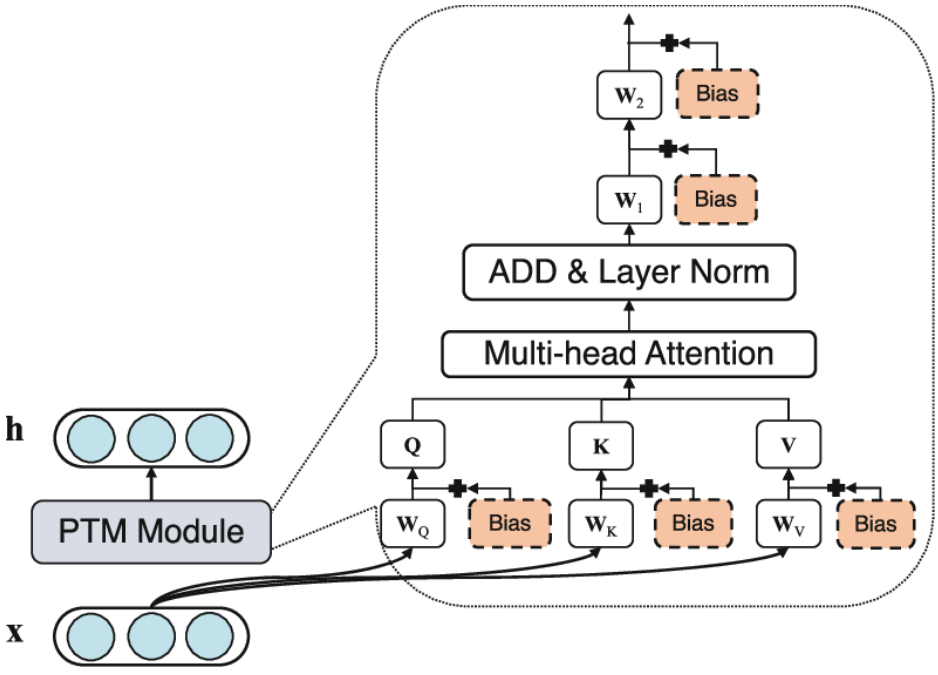
BitFit:

BitFit, PEFT yöntemleri arasında en minimal olanıdır. Bu yaklaşımda yalnızca bias terimleri güncellenir; diğer tüm parametreler sabit tutulur.

Parametre sayısı çok az güncellendiği için:

- Eğitim maliyeti son derece düşüktür
- Bellek kullanımı minimumdur
- Bazı görevlerde beklenenden iyi performans gösterebilir

Ancak karmaşık görevlerde LoRA veya Adapter kadar güçlü olmayabilir.



Genel Değerlendirme:

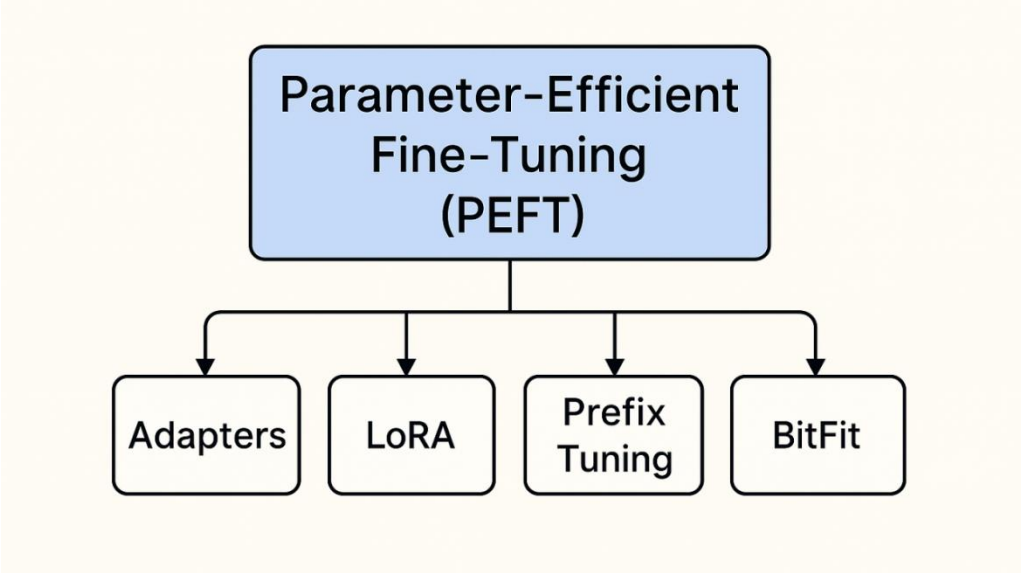
Full fine-tuning maksimum uyarlanabilirlik ve potansiyel performans sunarken yüksek maliyetlidir.

Feature extraction düşük maliyetli ancak sınırlı adaptasyon kapasitesine sahiptir.

PEFT yöntemleri ise bu iki yaklaşım arasında dengeli bir çözüm sunar.

LoRA, Prefix Tuning, Adapter ve BitFit gibi teknikler, modern büyük dil modellerinin verimli ve ölçeklenebilir şekilde güncellenmesini mümkün kılar.

Prefix Tuning, Adapter ve BitFit yöntemleri, Parameter-Efficient Fine-Tuning (PEFT) kapsamında yer alan ve modeli tamamen güncellemeden göreve uyarlamayı amaçlayan üç farklı yaklaşımdır.



Prefix Tuning, modelin attention katmanlarına öğrenilebilir "prefix" vektörleri ekleyerek model davranışını yönlendirir; ana ağırlıklar sabit kalır ve yalnızca bu ek bağlam parametreleri eğitilir.

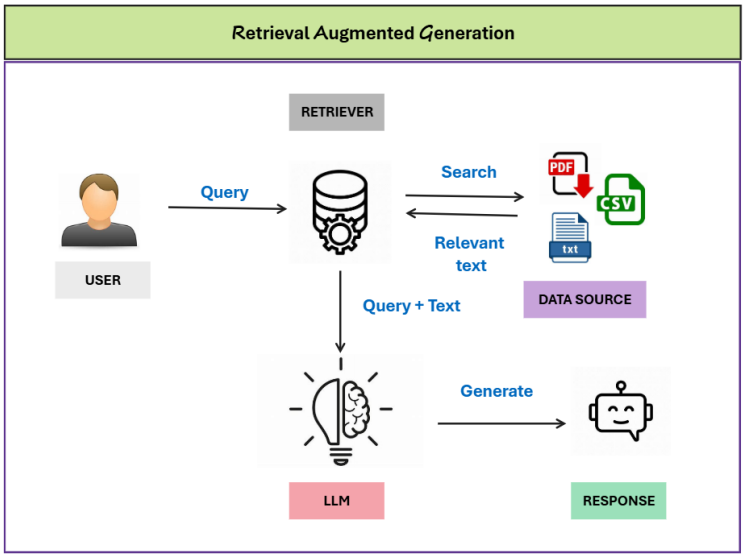
Adapter yönteminde ise transformer katmanlarının arasına küçük ve daraltılmış modüller eklenir; eğitim sürecinde sadece bu adapter katmanları güncellenir ve böylece modüler bir görev uyarlaması sağlanır.

BitFit yaklaşımı ise en minimal stratejidir; modelde yalnızca bias terimleri güncellenir, diğer tüm parametreler dondurulur.

Özetle Prefix Tuning bağlamsal yönlendirme, Adapter yapısal ek modülleme, BitFit ise en hafif parametre güncellemesi prensibiyle çalışır.

RAG Mantığı: Bilgi Çağırıp Cevap Üretme:

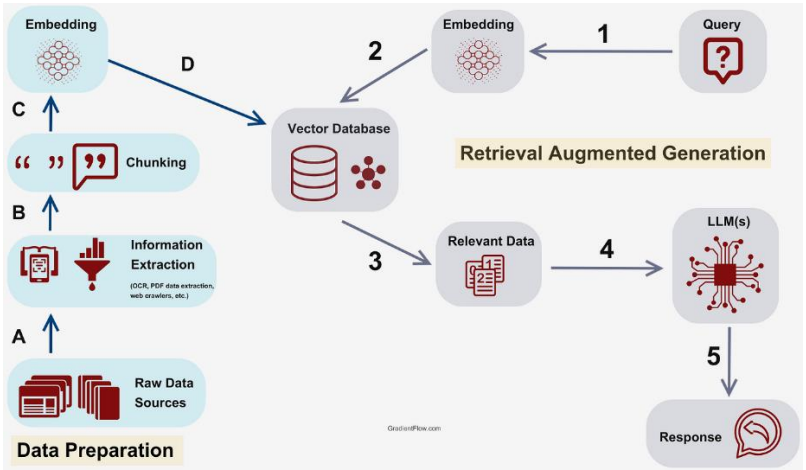
Retrieval-Augmented Generation (RAG), büyük dil modellerinin (LLM) yalnızca eğitim sırasında öğrendikleri parametre temsillerine dayanmak yerine, harici bilgi kaynaklarından veri çağırarak (retrieval) yanıt üretmesini sağlayan bir mimari yaklaşımdır. Bu yöntem, modelin içsel bilgisini dış veri kaynaklarıyla destekleyerek daha doğru, güncel ve doğrulanabilir çıktılar üretmesini amaçlar. Özellikle kurumsal bilgi tabanları, doküman arşivleri veya alan-özel (domain-specific) veri kümeleriyle çalışan sistemlerde RAG mimarisi kritik bir rol oynamaktadır.



RAG yaklaşımı üç temel aşamadan oluşur: Retrieval (Bilgi Çağırma), Augmentation (Bağlamı Genişletme) ve Generation (Yanıt Üretme).

1. Retrieval (Bilgi Çağırma)

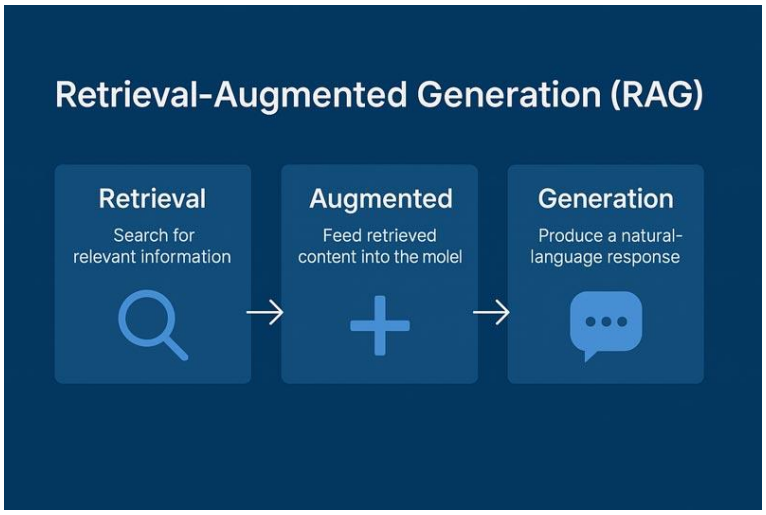
İlk aşamada, kullanıcıdan gelen sorgu bir embedding model aracılığıyla vektör temsiline dönüştürülür ve bu temsil bir vektör veritabanında (vector database) saklanan doküman embedding'leri ile karşılaştırılır. Bu işlem genellikle semantik arama (semantic search) yöntemleri kullanılarak gerçekleştirilir. Amaç, sorgu ile anlamsal olarak en ilişkili metin parçalarını (chunk) tespit etmektir.

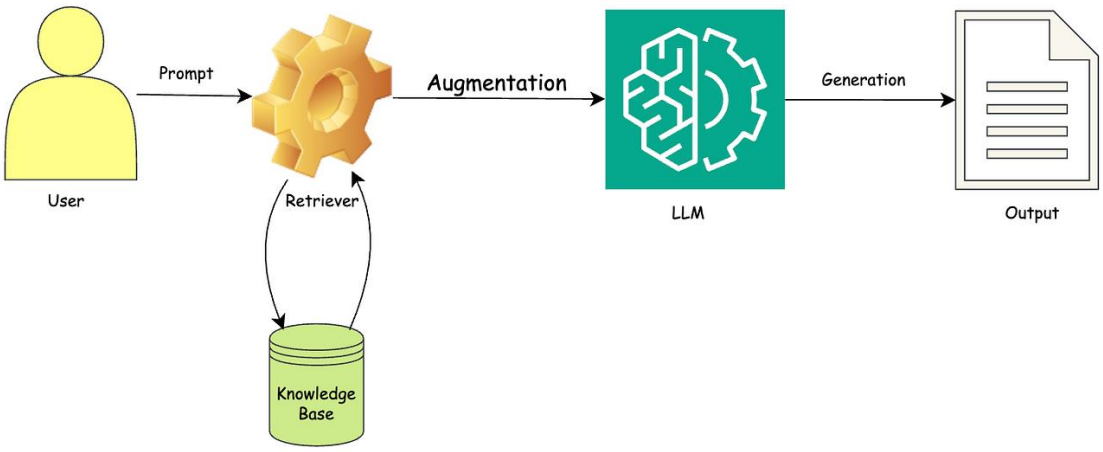


Kurumsal uygulamalarda bu veritabanı; şirket dokümanları, ürün katalogları, destek kayıtları veya özel meta verilerden oluşabilir. Bu sayede model, eğitim sürecinde görmediği ancak sistemde mevcut olan güncel veya özel bilgilere erişebilir. Retrieval aşamasının başarısı, embedding kalitesi, chunking stratejisi ve benzerlik ölçüm yöntemleri (örneğin cosine similarity) gibi teknik unsurlara bağlıdır.

2. Augmentation (Bağlamı Genişletme)

Retrieval aşamasında elde edilen ilgili metin parçaları, kullanıcı sorgusu ile birlikte yeniden yapılandırılarak genişletilmiş bir bağlam oluşturulur. Bu aşama, prompt engineering tekniklerinin devreye girdiği kritik bir adımdır. Amaç, çağrılan bilgilerin modele uygun bir formatta sunulması ve tekrar, bağlam kaybı veya anlamsal çakışma gibi sorunların minimize edilmesidir.

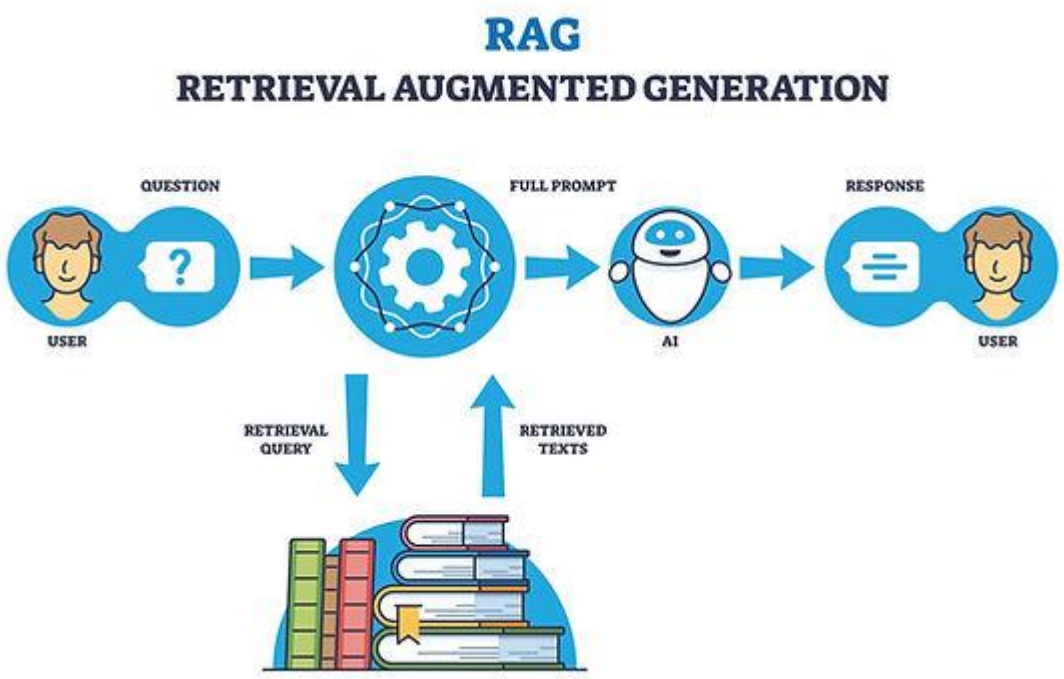


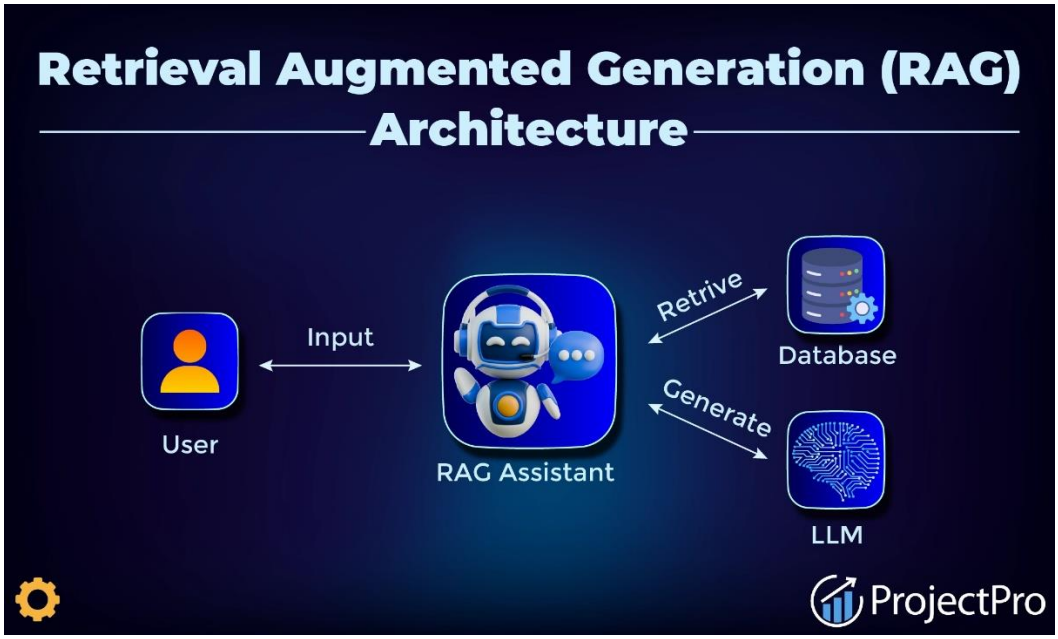


Bu genişletilmiş bağlam sayesinde model yalnızca kendi parametre bilgisini değil, aynı zamanda dış kaynaktan getirilen doğrulanabilir içerikleri de dikkate alarak yanıt üretir. Böylece bağlamsal sınırlamalar önemli ölçüde azaltılır.

3. Generation (Yanıt Üretme)

Son aşamada, genişletilmiş bağlam transformer tabanlı dil modeline iletilir. Model, hem kullanıcı sorgusuna hem de retrieval edilen içeriklere dayanarak yanıt üretir. Üretilen çıktı yalnızca modelin içsel bilgisine değil, aynı zamanda dış bilgi kaynaklarına da dayandığı için daha tutarlı ve güvenilir olma potansiyeline sahiptir.





Bu yapı, özellikle "halüsinasyon" olarak adlandırılan, modelin gerçek dışı ancak ikna edici görünen bilgiler üretme problemini azaltmada etkilidir. Çünkü model, cevabı üretirken somut ve referanslanabilir verilere dayanmaktadır.

RAG'ın Büyük Dil Modellerindeki Önemi:

Büyük dil modelleri geniş veri kümeleri üzerinde eğitilmiş olsa da, eğitim verileri belirli bir tarihe kadar olan bilgileri içerir ve genellikle kuruma özel veya alan-spesifik detayları kapsamaz. Bu durum, özellikle bankacılık, sigortacılık, sağlık veya hukuk gibi uzmanlık gerektiren sektörlerde doğruluk riskini artırır.

RAG mimarisi şu avantajları sağlar:

- Halüsinasyon Azaltma: Model, dış kaynaklardan doğrulanabilir bilgi aldığı için gerçek dışı üretim olasılığı azalır.
- Alan-Spesifik Doğruluk: Kurumsal veri tabanları entegre edilerek domain özelinde yüksek doğruluk sağlanır.
- Güncellik: Veri tabanı güncellenerek model yeniden eğitilmeden sistem güncel tutulabilir.
- Denetlenebilirlik: Yanıtın hangi dokümanlara dayandığı izlenebilir, böylece açıklanabilirlik artar.
- Maliyet Avantajı: Sürekli fine-tuning yapmak yerine veri tabanını güncellemek daha düşük maliyetlidir.

RAG Sistemlerinde Teknik Zorluklar:

RAG mimarisinde en kritik aşama retrieval sürecidir. Yanlış veya alakasız bağlam çağrılması durumunda, generation aşaması da hatalı sonuç üretebilir. Karşılaşılan başlıca teknik zorluklar şunlardır:

- Anlamsal belirsizlik (semantic ambiguity)
- Yüksek boyutlu vektör uzayında yakınlık problemleri
- Cosine similarity'nin yön odaklı değerlendirme yapması
- Chunk boyutu ve top-k parametrelerinin optimizasyonu
- Bağlam uzunluğu (context window) sınırlamaları

Sonuç:

Retrieval-Augmented Generation (RAG), büyük dil modellerinin yalnızca içsel parametre bilgisine dayalı üretim yapmasını aşarak, harici ve doğrulanabilir veri kaynaklarından beslenen hibrit bir üretim mimarisi sunmaktadır. Bu yaklaşım, doğruluk, güvenilirlik, güncellik ve denetlenebilirlik açısından önemli avantajlar sağlamaktadır. Özellikle alan-spesifik uygulamalarda ve kurumsal yapay zeka sistemlerinde RAG, ölçeklenebilir ve maliyet-etkin bir çözüm olarak öne çıkmaktadır.

LangChain ve Haystack kullanımı Nasıldır?

LangChain Nedir?

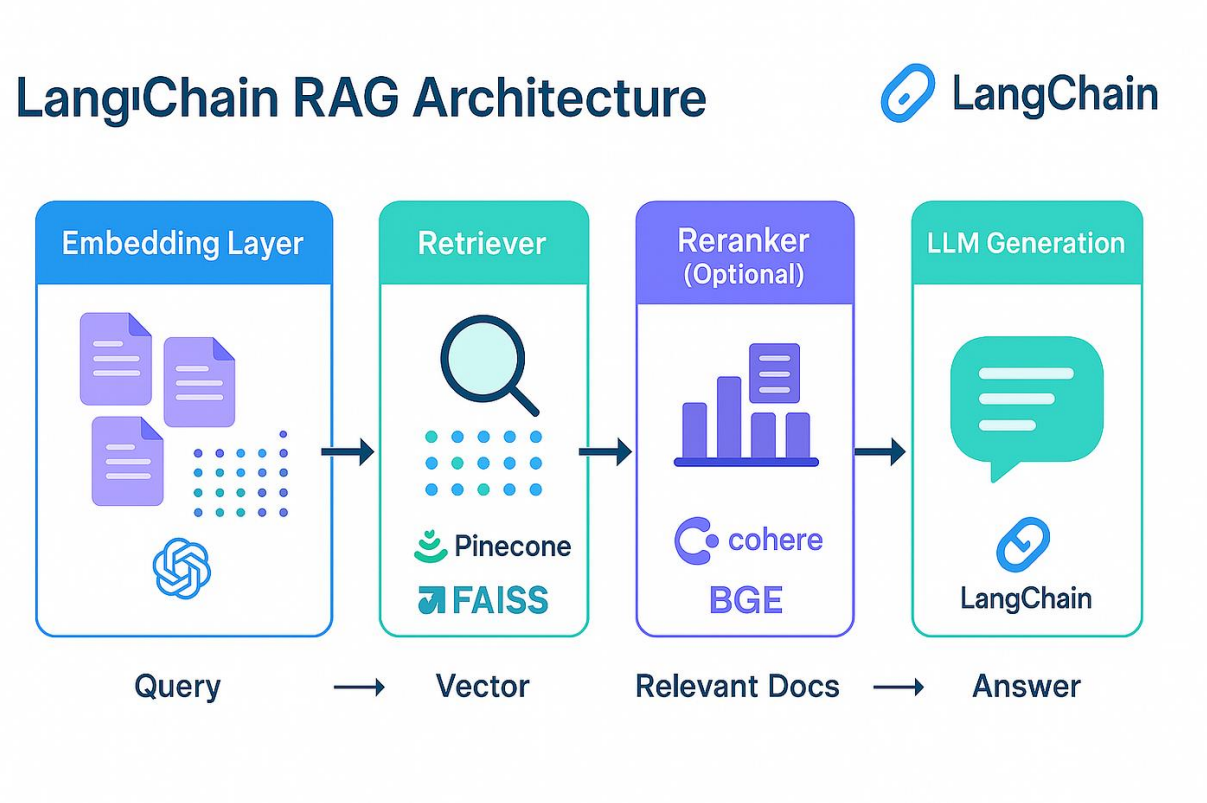
LangChain, büyük dil modelleri (LLM) ile çalışmayı daha modüler, esnek ve güçlü hâle getiren bir açık kaynak yazılım çerçevesidir. LangChain'in temel hedefi, sadece modelden yanıt almakla kalmayıp modelin dış kaynaklardan bilgi çekmesini, adımlı işlem zincirleri (chains) kurmasını ve gerçek uygulamalarda üretken yapay zekâ çözümleri geliştirmeyi kolaylaştırmaktır.

LangChain Ne İçin Kullanılır?

LangChain genellikle şu amaçlarla kullanılır:

- RAG (Retrieval-Augmented Generation) sistemleri oluşturma
- Birden fazla adımı zincir hâline getirerek karmaşık görevleri çözme
- Araçlar (tools) ile etkileşime giren agent'lar geliştirme
- Prompt'ları dinamik biçimde yönetme ve optimize etme

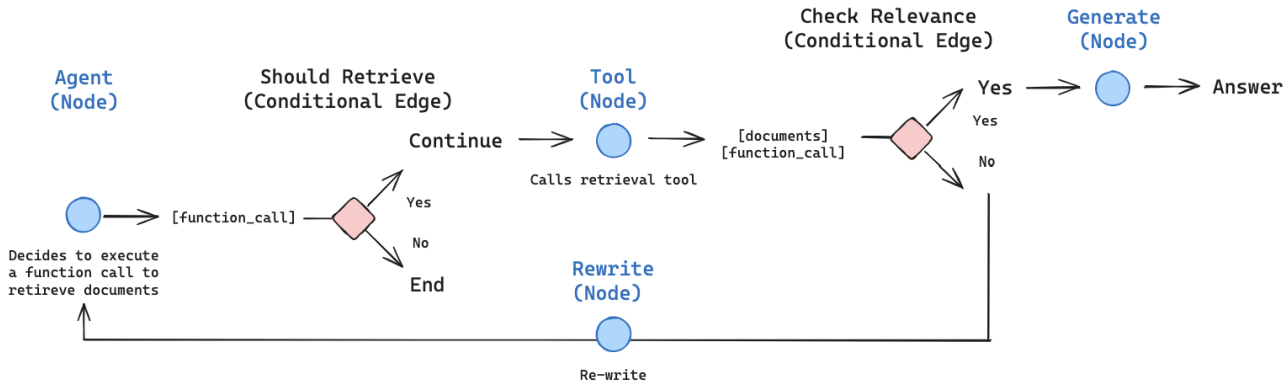
LangChain, sadece LLM çağırmakla kalmaz; uygulama mantığını modelin üzerine kurmanız için yapı blokları sunar.



Nasıl Çalışır?

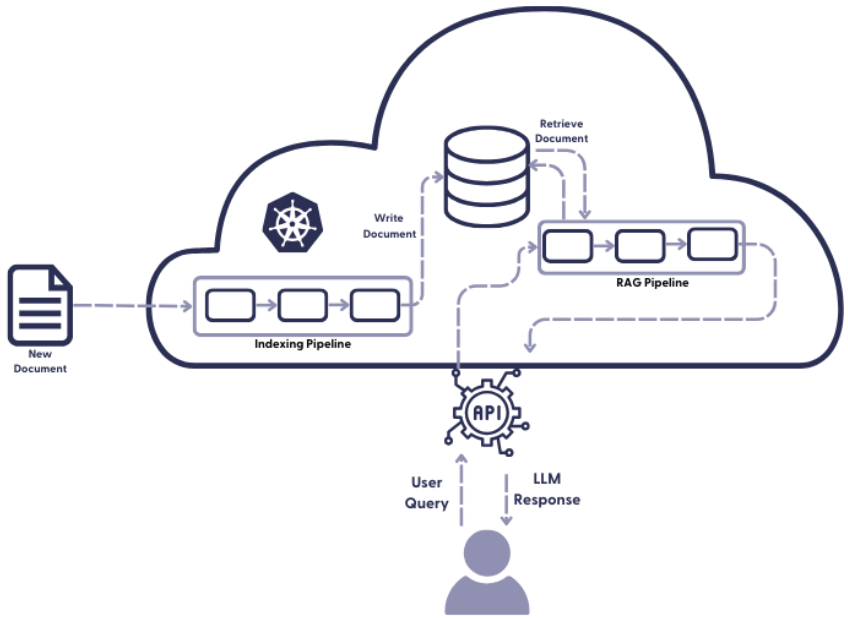
LangChain, temel olarak şu bileşenleri içerir:

1. LLM Wrapper'ları: Farklı dil modellerini tek bir standart arayüzle kullanmaya yarar.
2. Chains (Zincirler): Birden fazla adımı sıra ile çalıştırarak daha karmaşık görevleri çözer.
3. Agents (Temsilciler): Modelin araçlarla (örneğin web araması, veri tabanı sorgu motoru) etkileşime geçtiği sistemler.
4. Retrievers & Vector Stores: RAG gibi dış bilgi kaynaklarından veri çağırmayı sağlar.



Haystack Nedir?

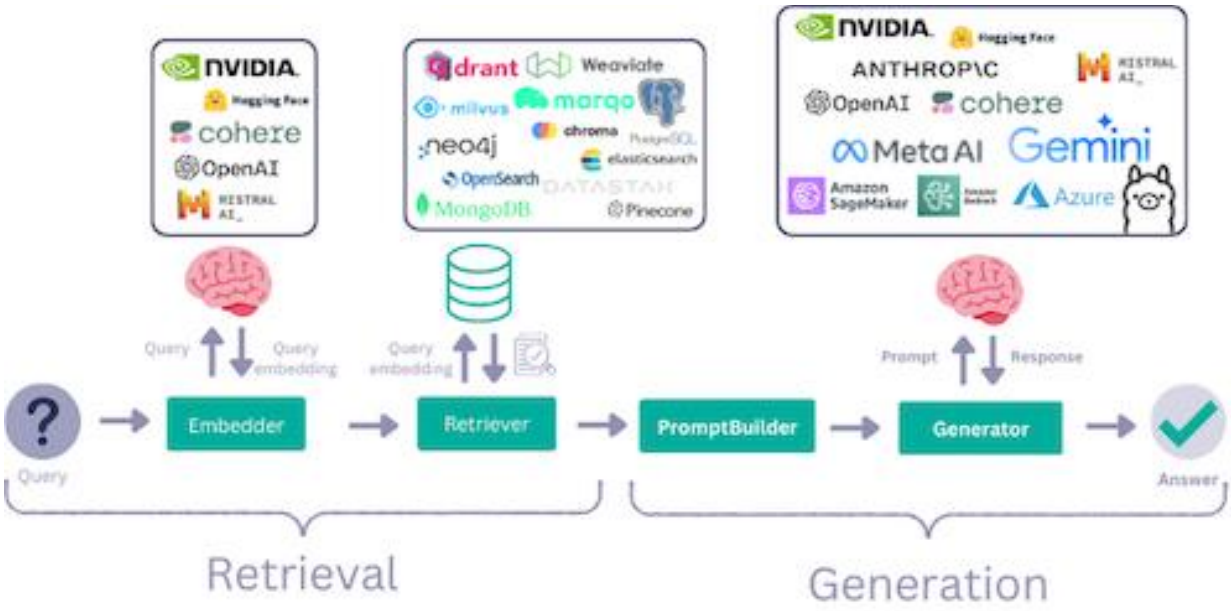
Haystack, açık kaynaklı bir document-centric (doküman odaklı) bilgi erişim ve arama framework'üdür. Özellikle metin veri tabanı üzerinde hızlı arama, retrieval ve NLP tabanlı çıkarım işleri yaparken kullanılır. Haystack, RAG benzeri uygulamaları daha esnek ve ölçeklenebilir hâle getirmeye odaklanır.



Haystack Ne İçin Kullanılır?

Haystack özellikle aşağıdaki kullanım senaryolarında tercih edilir:

- Kurumsal dokümanlar üzerinde soru-yanıt (QA) sistemleri
- RAG pipeline'ları (Retriever + Reader)
- Elasticsearch / Milvus / Weaviate gibi vektör veri tabanlarıyla entegre arama
- Doküman indeksleme ve parçalama (chunking)



Haystack, büyük koleksiyonlardaki belgeleri vektörleştirmek, bunları veritabanında saklamak ve modelin bu belgelerden bilgi çağırmasını kolaylaştırmak için yapı taşları sunar.

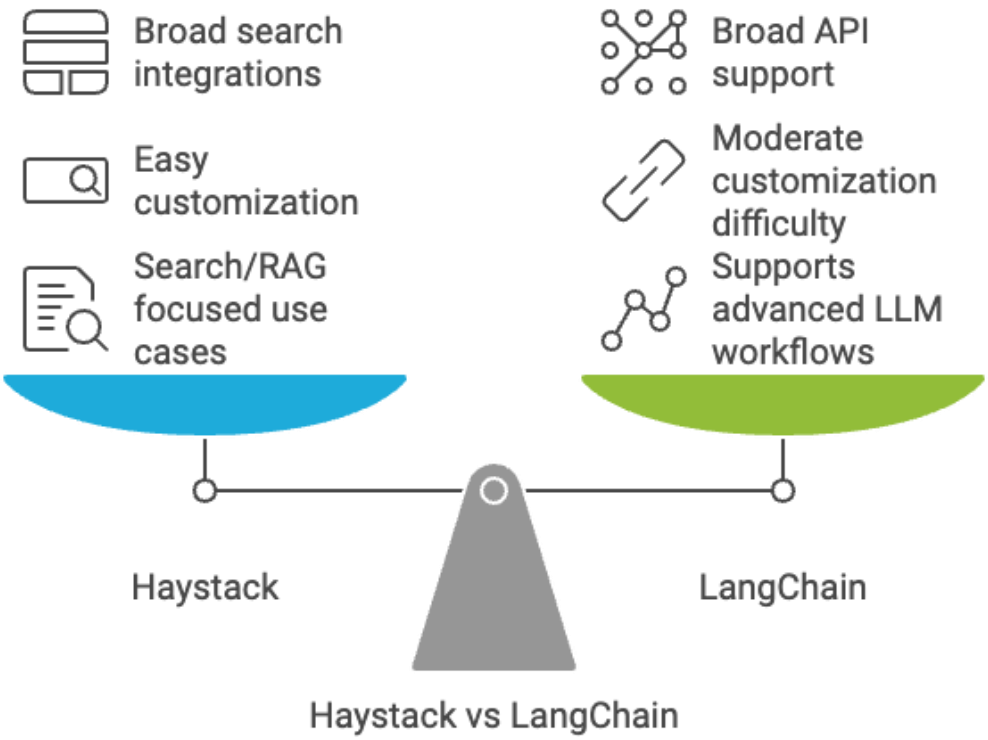
Nasıl Çalışır?

Haystack'in temel bileşenleri şunlardır:

1. Document Store: Verilerin saklandığı yer (Elasticsearch, FAISS, Weaviate vb).
2. Retriever: Sorguya göre en ilişkili belge parçalarını seçer.
3. Reader: Retriever'ın döndürdüğü parçalardan yararlanarak detaylı yanıt üretir.
4. Pipeline: Bu bileşenlerin bir iş akışı içinde birleştiği yapı.

LangChain ile Haystack Arasındaki Fark ve Kullanım Amaçları:

Özellik	LangChain	Haystack
Odak	LLM odaklı workflow / agent & tools	Doküman arama & QA pipeline
RAG içinde rol	Retrievers, tools, agent'lar	Gömülü retriever + reader
Tipik kullanım	Karmaşık LLM çözümleri, agent tabanlı sistemler	Belge tabanlı soru-yanıt ve arama
Vektör database	Evet (LangChain araçlarla)	Evet (FAISS, Milvus, Weaviate vb)



LangChain, LLM etrafında daha yüksek seviyeli çözümler (örneğin "model + web araması + database + kod yazma agent'ı") geliştirmek için güçlü bir çerçevedir.

Haystack ise daha çok büyük doküman koleksiyonları üzerinde doğru bilgi çağırma (retrieval) ve kesin yanıt üretme (reader) görevlerine odaklanır.

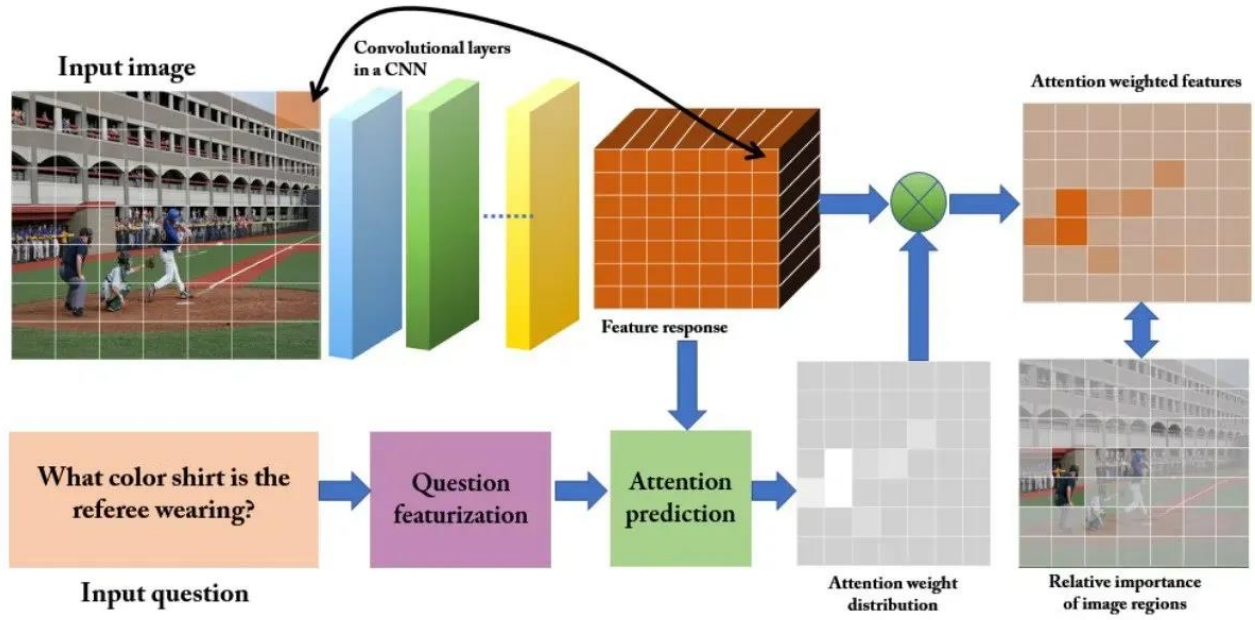
VQA ve Image Captioning Üzerinden RAG Uygulamasının Farkı Nedir?

VQA ve Image Captioning Üzerinden RAG Uygulamasının Farkı:

Görsel tabanlı yapay zeka sistemlerinde Retrieval-Augmented Generation (RAG) yaklaşımı yalnızca metin verisiyle sınırlı değildir; görsel–metin birleşimli (multimodal) sistemlerde de uygulanabilmektedir. Bu bağlamda iki önemli görev öne çıkar: Visual Question Answering (VQA) ve Image Captioning. Her iki görev de görsel girdiye dayansa da, RAG mimarisinin bu iki görevdeki rolü, bilgi ihtiyacı ve üretim stratejisi açısından önemli farklılıklar göstermektedir.

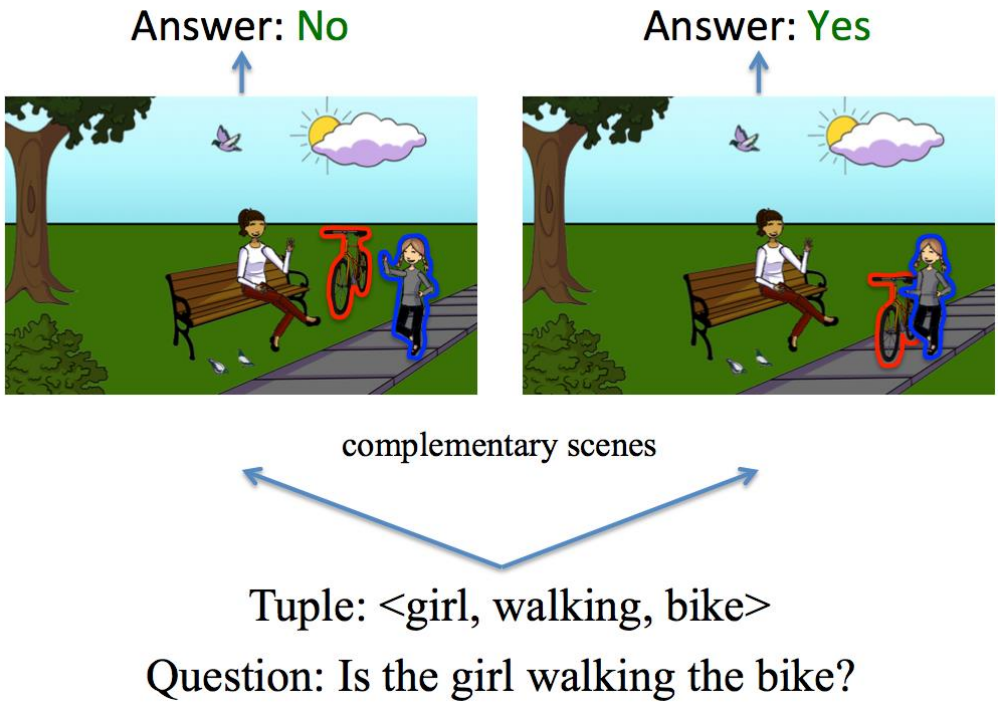
1. VQA (Visual Question Answering) Nedir?

VQA, bir görsel ve o görsele ilişkin bir soru verildiğinde, sistemin doğru ve bağlama uygun bir yanıt üretmesini amaçlayan bir multimodal görevdir.



Bu sistemlerde model, hem görsel özellikleri (vision encoder aracılığıyla) hem de metinsel sorguyu birlikte işler.

Model yalnızca görselden çıkarım yapabilir; ancak bazı durumlarda görsel tek başına yeterli bilgi sunmayabilir.



Örneğin:

- Görsel: Bir hastane ortamı
- Soru: “Bu ortamda hangi tıbbi ekipmanlar bulunuyor?”

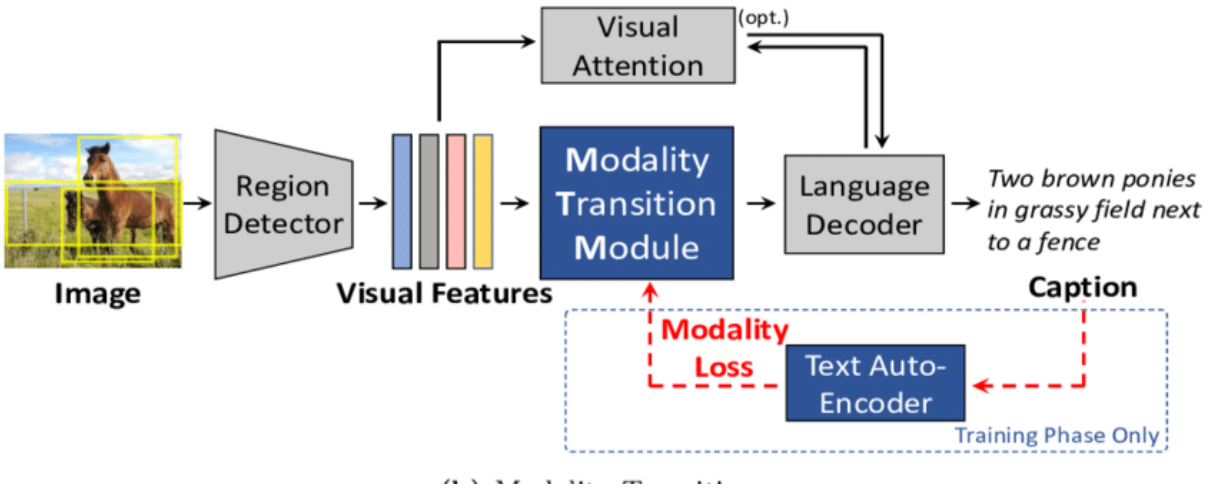
Model yalnızca görselden çıkarım yapabilir; ancak bazı durumlarda görsel tek başına yeterli bilgi sunmayabilir. Özellikle teknik, alan-spesifik veya harici bilgi gerektiren sorularda RAG entegrasyonu kritik hale gelir.

2. Image Captioning Nedir?

Image Captioning, bir görsel için otomatik olarak açıklayıcı bir metin üretme görevini ifade eder. Bu görevde kullanıcıdan ek bir soru gelmez; model doğrudan görsel içeriği betimleyen bir açıklama üretir.

Örneğin:

- Görsel: Sahilde oynayan çocuklar
- Üretilen çıktı: “Gün batımında sahilde oynayan iki çocuk.”



Captioning sistemleri genellikle görselden genel bir sahne açıklaması üretir ve çoğu durumda dış bilgiye ihtiyaç duymaz. Ancak daha detaylı, teknik veya bağlamsal açıklamalar gerektiğinde RAG devreye girebilir.

VQA’da RAG Kullanımı:

VQA sistemlerinde RAG, özellikle görselin ötesinde bilgi gerektiren sorularda kullanılır. Burada süreç şu şekilde işler:

1. Görsel, bir vision encoder (örneğin CLIP veya ViT) aracılığıyla embedding’e dönüştürülür.
2. Kullanıcı sorusu metinsel embedding’e çevrilir.
3. Görsel + soru temsili, harici bir bilgi tabanında arama yapmak için kullanılır.
4. Retrieval edilen belgeler genişletilmiş bağlam olarak modele sunulur.
5. Model, hem görsel hem harici bilgiye dayanarak yanıt üretir.

Örneğin:

- Görsel: Tarihi bir bina
- Soru: “Bu bina hangi döneme aittir?”

Model yalnızca görselden mimari stil tahmini yapabilir; ancak RAG ile tarihi bina veri tabanından bilgi çekerek daha doğru ve doğrulanabilir yanıt verebilir.

VQA'da RAG'ın Avantajı

- Görselin ötesinde bilgi sağlar
- Alan-spesifik doğruluğu artırır
- Halüsinasyon riskini azaltır
- Özellikle medikal veya teknik alanlarda kritik rol oynar

Image Captioning'de RAG Kullanımı:

Image captioning görevinde RAG kullanımı daha farklı bir motivasyona dayanır. Burada amaç genellikle:

- Daha zengin açıklamalar üretmek
- Görsele dair arka plan bilgisi eklemek
- Domain-spesifik detayları dahil etmek

Örneğin:

- Görsel: Bir sanat eseri
- Basit caption: "Bir tablo."
- RAG destekli caption: "Vincent van Gogh'un 1889 tarihli Yıldızlı Gece tablosu."

Bu durumda sistem:

1. Görseli analiz eder.
2. Görsel embedding'i üzerinden sanat veri tabanında arama yapar.
3. Elde edilen metadata'yı caption üretiminde kullanır.

Captioning'de RAG'ın Rolü:

- Açıklamayı daha bilgilendirici hale getirir
- Görselin bağlamını genişletir
- Kültürel, tarihi veya teknik detay ekler

VQA ve Captioning Arasındaki RAG Farkı:

RAG'in bu iki görevdeki temel farkı, bilgi ihtiyacının doğasından kaynaklanır:

Özellik	VQA + RAG	Captioning + RAG
Girdi	Görsel + Soru	Sadece Görsel
Amaç	Soruya doğru yanıt	Görseli açıklamak
Retrieval tetikleyici	Soru	Görsel içeriği
Bilgi ihtiyacı	Çoğu zaman zorunlu	Genellikle isteğe bağlı
Kritik kullanım alanı	Medikal, teknik, QA sistemleri	Sanat, kültür, e-ticaret

VQA'da RAG çoğu zaman gereklidir çünkü soru modelin eğitim bilgisini aşabilir. Captioning'de ise RAG daha çok açıklamayı zenginleştirmek için kullanılır.

Sonuç:

VQA ve image captioning görevleri her ne kadar görsel tabanlı üretim sistemleri olsa da, RAG entegrasyonunun işlevi ve önemi farklıdır.

VQA'da RAG genellikle doğruluk ve bilgi tamamlama amacıyla zorunlu bir bileşen hâline gelirken,

image captioning'de daha çok açıklamayı zenginleştiren ve bağlamsal derinlik kazandıran bir mekanizma olarak kullanılır.

Multimodal RAG yaklaşımları, görsel yapay zekâ sistemlerinin yalnızca gördüğünü betimlemesini değil, aynı zamanda gördüğü hakkında bilgi üretmesini mümkün kılmaktadır.

KAYNAKLAR:

- https://www.ai21.com/glossary/foundational-llm/fine-tuning/#:~:text=Fine%2Dtuning%20is%20the%20process,summarization%20and%20customer%20service%20automation.
- https://www.geeksforgeeks.org/deep-learning/what-is-fine-tuning/
- https://mehmetozkaya.medium.com/fine-tuning-why-and-when-to-use-a2fa462f7a15
- https://www.komtas.com/en/glossary/fine-tuning-nedir
- https://medium.com/@lmpo/mastering-classification-metrics-a-deep-dive-into-accuracy-precision-recall-f1-score-and-f8caaf669bf0
- https://apxml.com/courses/cnns-for-computer-vision/chapter-6-advanced-transfer-learning-domain-adaptation/fine-tuning-feature-extraction-advanced
- https://codefinity.com/courses/v2/9c23bf60-276c-4989-a9d7-3091716b4507/f2b3e2f0-2c36-4612-b609-3f72afdda599/4f088c2c-2080-4113-a3d2-4f9f4357549e
- https://www.researchgate.net/figure/Feature-extractor-and-classifier-model_fig2_365645939
- https://daehnhardt.com/blog/2022/04/06/tensorflow-transfer-learning-image-classification-fine-tuning-data-augmentation-predictive-modeling-image-classification/
- https://medium.com/data-reply-it-datatech/transfer-learning-feature-extraction-and-fine-tuning-db7d82767992
- https://www.ibm.com/think/topics/parameter-efficient-fine-tuning#:~:text=IBM%20Think-,What%20is%20parameter%2Defficient%20fine%2Dtuning%20(PEFT)%3F,specific%20tasks%20or%20data%20sets.
- https://www.leewayhertz.com/parameter-efficient-fine-tuning/
- https://pub.towardsai.net/low-rank-adaptation-lora-fedf37b92026
- https://lightning.ai/pages/community/article/understanding-llama-adapters/
- https://www.researchgate.net/figure/Structure-of-prefix-tuning-The-parameters-of-the-layer-inside-the-dotted-line-box-can_fig5_372296173
- https://www.researchgate.net/figure/The-illustration-of-the-BitFit-method_fig10_373358021
- https://medium.com/nane-limon/b%C3%BCy%C3%BCk-dil-modellerinin-s%C4%B1n%C4%B1rlar%C4%B1n%C4%B1-a%C5%9Fmak-rag-ile-bilgi-geni%C5%9Fletme-sanat%C4%B1-part-1-1a930b42402f
- https://pub.towardsai.net/what-is-rag-a-clear-and-simple-explanation-a3980bd3a61a
- https://medium.com/enterprise-rag/an-introduction-to-rag-and-simple-complex-rag-9c3aa9bd017b
- https://pub.towardsai.net/rag-explained-a-comprehensive-guide-to-mastering-retrieval-augmented-generation-e3f9b46ac06b
- https://conversed.ai/what-is-rag-retrieval-augmented-generation/
- https://www.minervacq.com/post/make-customer-service-reps-smarter-with-rag-retrieval-augmented-generation

- https://www.projectpro.io/article/rag-architecture/1079
- https://medium.com/@bhagyarana80/langchain-rag-architecture-explained-from-embedding-to-generation-e267610c8180
- https://haystack.deepset.ai/tutorials/35_evaluating_rag_pipelines
- https://www.makewithdata.tech/p/is-haystack-better-than-langchain
- https://visualqa.org/
- https://blog.roboflow.com/what-is-vqa/
- https://www.projectpro.io/article/image-captioning-deep-learning-project/717

Hazırlayan: Rabia Güllü KOCAEL