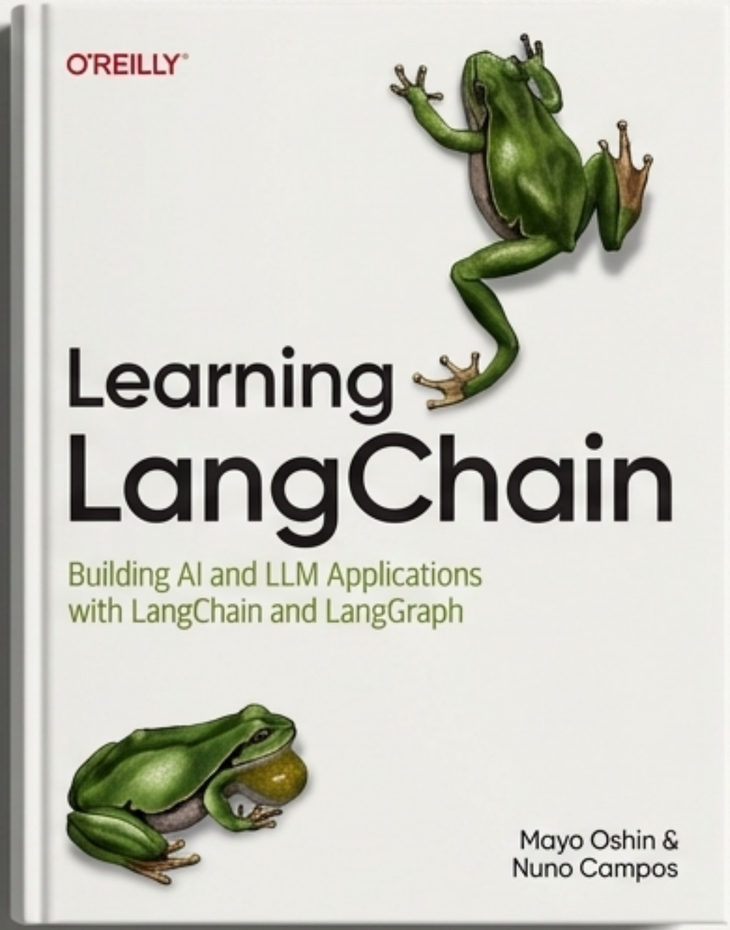


Fikirden Üretime: LangChain ile Akıllı AI Agent'lar Geliştirme Yolculuğu

LangChain ve LangGraph kullanarak üretime hazır, karmaşık AI uygulamaları oluşturmak için pratik bir rehber.

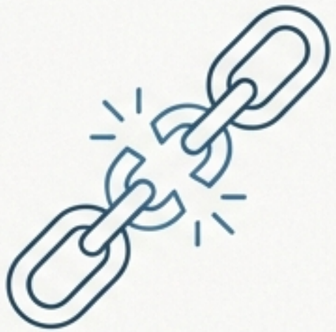
Bu sunum, temel bir LLM uygulamasından başlayarak, kendi verilerinizle konuşabilen, geçmiş konuşmaları hatırlayabilen ve otonom olarak eyleme geçebilen karmaşık bir AI agent'a uzanan geliştirici yolculuğunu adım adım anlatmaktadır. LangChain'in temel yapı taşlarından başlayıp LangGraph ile gelişmiş mimariler kurmaya ve LangSmith ile uygulamayı üretime taşımaya kadar tüm süreci kapsayacağız.

LLM'ler Güçlüdür, Ancak Sınırları Vardır

ChatGPT'nin ortaya çıkışıyla birlikte üretken yapay zeka uygulamaları geliştirmek hiç olmadığı kadar kolaylaştı. Ancak LLM'leri prototipten üretime taşımak, temel bazı zorlukları beraberinde getirir:

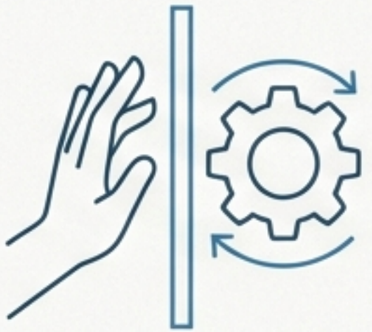
Bilgi Kesintisi (Knowledge Cutoff)

LLM'ler, eğitim verilerinin ötesindeki güncel olaylar veya özel bilgiler hakkında bilgi sahibi değildir. Bu durum, yanlış veya "halüsinasyon" olarak adlandırılan yanıltıcı çıktılara yol açabilir.



Durumsuzluk (Statelessness)

LLM'ler doğası gereği durumsuzdur. Her bir etkileşimi bağımsız olarak işlerler ve önceki konuşmaları hatırlamazlar. Bu, doğal bir diyalog akışı oluşturmayı zorlaştırır.



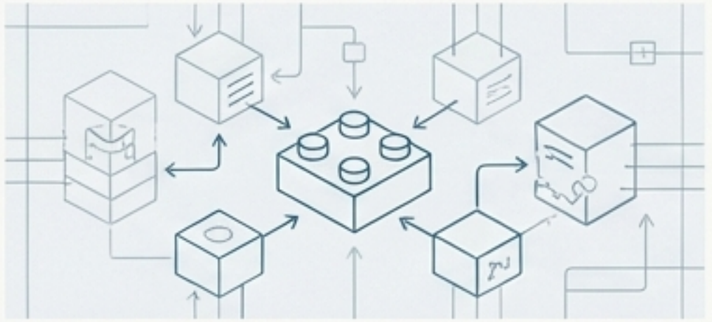
Eyleme Geçememe (Inability to Act)

LLM'ler metin üretirler, ancak dış dünyayla etkileşime geçemez, API'ları çağıramaz veya bir hesap makinesi gibi basit araçları bile kullanamazlar.

Gerçek dünyada değer yaratan uygulamalar, bu sınırları aşmak zorundadır.

Yol Haritanız: LangChain ve Ekosistemi

LangChain, LLM'lerin bu temel sınırlılıklarını aşmak için tasarlanmış açık kaynaklı bir framework'tür. En ilginç LLM uygulamalarının, LLM'leri 'diğer hesaplama veya bilgi kaynakları' ile birleştirmesi gerektiği fikrinden doğmuştur.

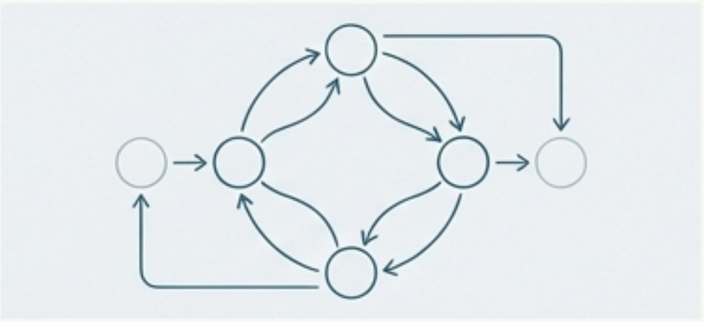


LangChain

Uygulama Geliştirme

LLM uygulamaları için değiştirilebilir yapı taşları sunar: LLM entegrasyonları, Prompt şablonları, RAG bileşenleri (Document Loader, Text Splitter, Vector Store), Araçlar (Tools) ve daha fazlası.

Uygulamanızı sağlayıcıdan bağımsız ve geleceğe hazır hale getirir.



LangGraph

Agent Mimarileri Oluşturma

Döngüler ve durum yönetimi içeren karmaşık, çok adımlı AI agent'lar oluşturmak için bir kütüphanedir. Hafıza eklemek ve LLM'e karar verme yeteneği kazandırmak için kullanılır.

Durumlu (stateful) ve otonom sistemler inşa etmenizi sağlar.

LangSmith

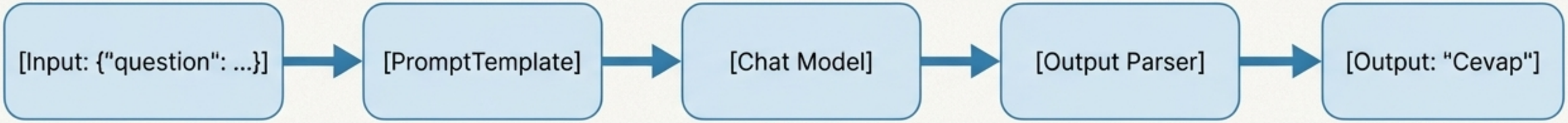
Hata Ayıklama, Test ve Gözlemleme

AI iş akışlarınızı uçtan uca izlemek, test etmek, değerlendirmek ve monitör etmek için tasarlanmış bir platformdur.

Uygulamanızın güvenilirliğini ve performansını artırır.

Yolculuğun İlk Adımı: Temel Bir LLM Zinciri Oluşturmak

Her şey basit bir zincirle başlar. LangChain Expression Language (LCEL) kullanarak, farklı bileşenleri birbirine kolayca bağlayabiliriz. Bu, LLM uygulamasının en temel formudur.



PromptTemplate: Kullanıcı girdisini dinamik olarak bir prompt'a yerleştirmeyi sağlar. Tekrar kullanılabilir ve yönetimi kolaydır.

Chat Model (LLM): OpenAI, Anthropic, Google gibi farklı sağlayıcılarla ortak bir arayüz üzerinden etkileşim kurar.

Output Parser: LLM'in metin çıktısını JSON veya CSV gibi yapılandırılmış formatlara dönüştürür.

Python Kodu:

```
chain = prompt | model | parser
```

JavaScript Kodu:

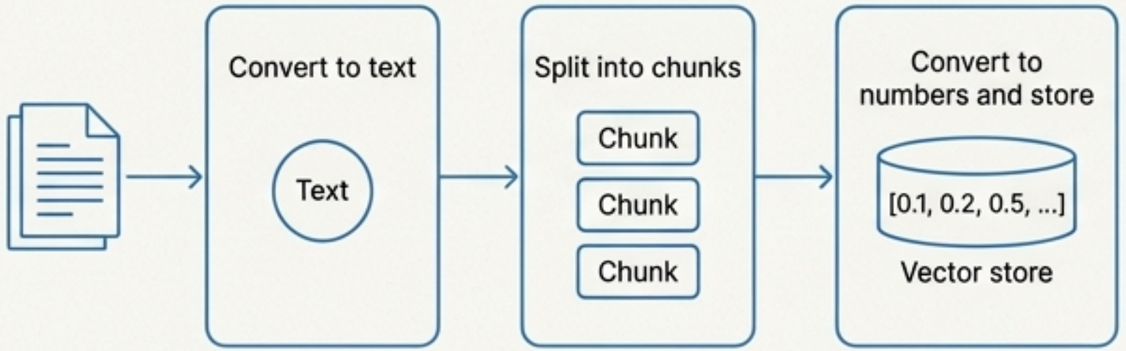
```
const chain = prompt.pipe(model).pipe(parser)
```

Bu yapı, LLM'in yeteneklerini programatik olarak kullanmanın temelini oluşturur. Ancak henüz kendi verilerimizle konuşamıyor.

Yetenek Geliştirme: Verilerinizle Konuşan Uygulamalar Yaratmak (RAG)

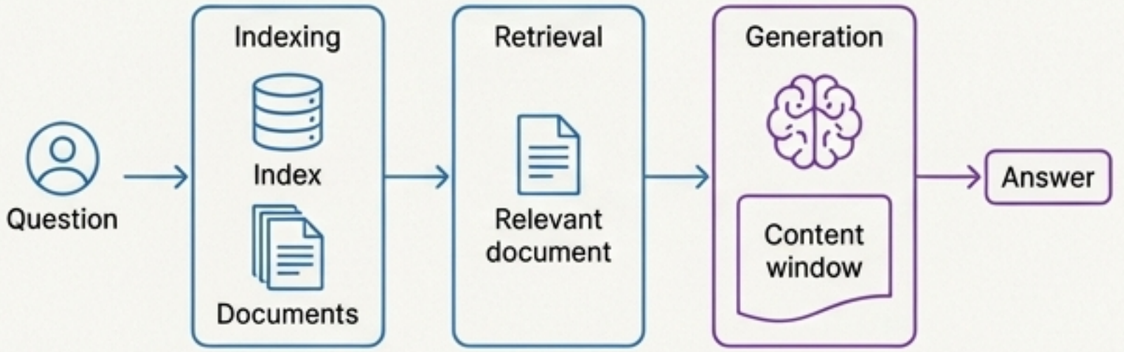
Problem: Temel LLM zincirimiz, yalnızca eğitim verilerinde bulunan bilgilere erişebilir. Peki ya şirketinizin özel dokümanları veya güncel veriler hakkında soruları nasıl yanıtlayacak?

Çözüm: Retrieval-Augmented Generation (RAG)
RAG, LLM'i harici bir bilgi tabanına bağlayarak bu sorunu çözer. Süreç, kullanıcı sorusuna en uygun bilgiyi bulup LLM'e "bağlam" olarak sunarak çalışır.



Aşama 1: Dizinleme (Indexing)

Dokümanlarınız metne dönüştürülür, anlamlı parçalara ayrılır, sayısal temsillere (embedding) çevrilir ve hızlı arama için bir vektör veritabanında saklanır.

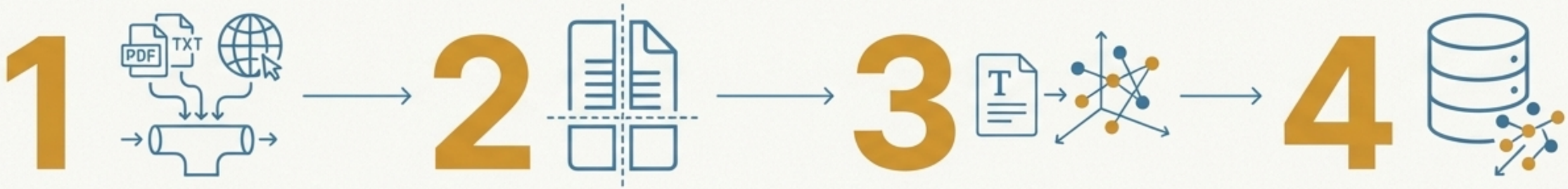


Aşama 2: Geri Getirme ve Üretme (Retrieval & Generation)

Kullanıcının sorusu da bir embedding'e dönüştürülür ve veritabanındaki en ilgili doküman parçalarını bulmak için kullanılır. Bu parçalar, doğru bir cevap üretmesi için LLM'e bağlam olarak verilir.

RAG Adım 1: Bilgi Tabanınızı Hazırlama (Dizinleme)

Etkili bir RAG sistemi kurmanın ilk adımı, verilerinizi LLM'in anlayabileceği ve arayabileceği bir formata dönüştürmektir. Bu sürece "ingestion" veya "indexing" denir.



Yükleme (Load)

PDF, TXT, HTML, hatta Notion veya Slack gibi kaynaklardan metin verilerini çıkarmak.

LangChain Aracı: `Document Loaders` (PyPDFLoader, WebBaseLoader vb.)

Bölme (Split)

Uzun dokümanları, anlamsal bütünlüğü koruyarak daha küçük, yönetilebilir parçalara (chunks) ayırmak.

LangChain Aracı: `Text Splitters` (RecursiveCharacterTextSplitter)

Gömme (Embed)

Her metin parçasının anlamsal içeriğini, 'embedding' adı verilen yoğun bir sayısal vektöre dönüştürmek. Benzer anlamlara sahip metinler, vektör uzayında birbirine daha yakın olur.

LangChain Aracı: `Embedding Models` (OpenAIEmbeddings, Cohere, vb.)

Depolama (Store)

Bu vektörleri, anlamsal benzerlik aramalarını verimli bir şekilde yapabilen özel bir veritabanında saklamak.

LangChain Aracı: `Vector Stores` (PGVector, Weaviate, vb.)

RAG Adım 2: Doğru Bilgiyi Akıllıca Geri Getirme (Retrieval)

Verileriniz dizinlendiğinde, bir sonraki adım, kullanıcının sorusuna en uygun bağlamı bulup LLM'e sunmaktır. Ancak ham kullanıcı sorguları genellikle yetersiz kalır.

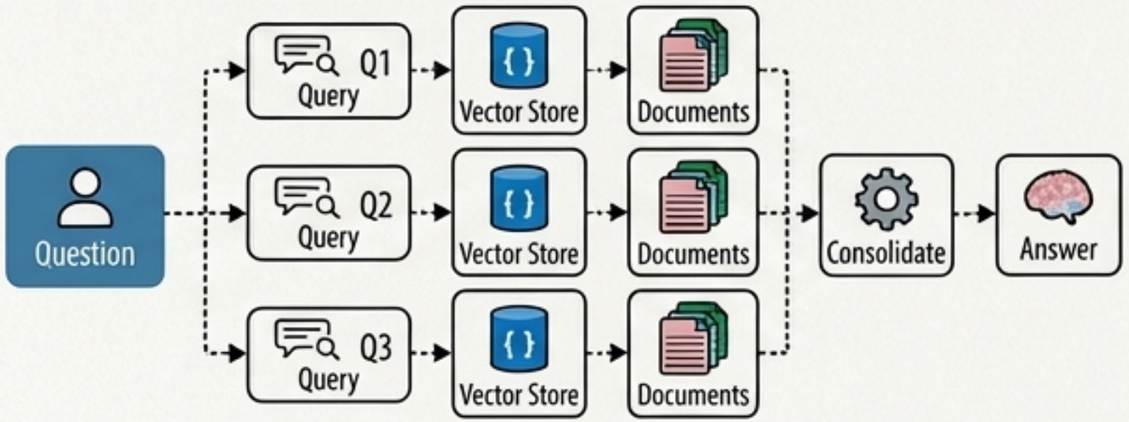
Gelişmiş Retrieval Stratejileri

Problem: Kullanıcı sorguları belirsiz, çok geniş veya alakasız bilgiler içerebilir. Bu, yanlış dokümanların getirilmesine ve dolayısıyla yanlış yanıtlara yol açar.

Çözüm: Sorgu Dönüşümü (Query Transformation) - Kullanıcının orijinal sorgusunu, daha iyi arama sonuçları elde etmek için LLM yardımıyla yeniden yazmak veya genişletmek.

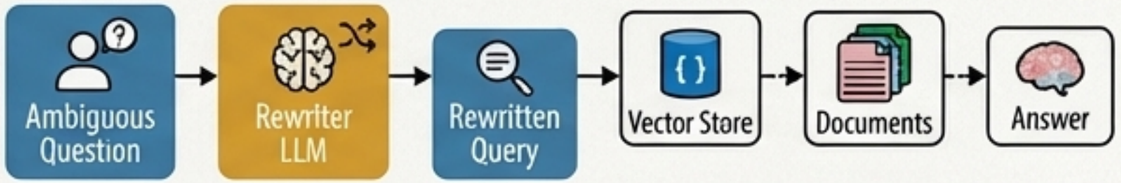
1. Multi-Query Retrieval

Tek bir kullanıcı sorusundan birden fazla, farklı açılardan arama sorgusu üretmek. Bu, daha kapsamlı ve çeşitli dokümanların bulunmasını sağlar. Sonuçlar birleştirilerek LLM'e sunulur.



2. Rewrite-Retrieve-Read

LLM'den, alakasız detayları temizleyerek veya belirsizliği gidererek, kullanıcıan 'gerçek niyetini' yansıtan daha net bir arama sorgusu oluşturmasını istemek.



Etkili retrieval, sadece vektör araması yapmak değil, aynı zamanda doğru soruyu sormaktır.

Seviye Atlama: Durum ve Hafıza Yönetmi (LangGraph)

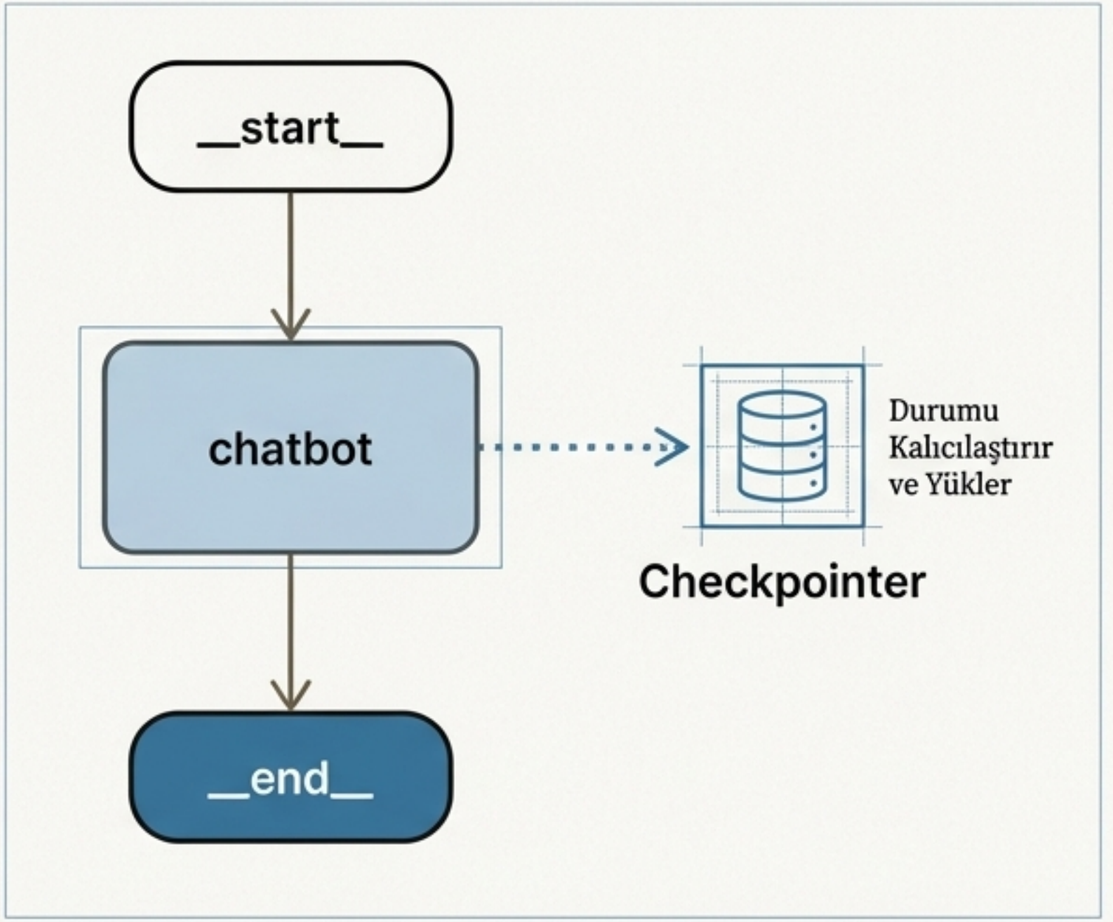
Problem: RAG uygulamamız artık verilerimizle konuşabiliyor, ancak her konuşma sıfırdan başlıyor. LLM'ler durumsuzdur (stateless). Önceki etkileşimleri nasıl hatırlayabilir ve çok adımlı görevleri nasıl yürütebilir?

Çözüm: LangGraph ile Durumlu Mimariler Oluşturma

LangGraph, durumu (state) açıkça yöneten, döngüler içeren ve çok adımlı mantığı uygulayan uygulamalar oluşturmak için bir kütüphanedir. Geleneksel "zincirlerin" ötesine geçerek "graflar" oluşturmamızı sağlar.

LangGraph'in Temel Kavramları

- **State:** Grafiğin hafızası. Uygulama çalışırken güncellenen ve tüm adımlar arasında paylaşılan veri yapısı.
- **Nodes:** İş birimleri. Mevcut durumu girdi olarak alan ve bir güncelleme döndüren fonksiyonlar (örneğin, bir LLM'i çağırmak, bir aracı çalıştırmak).
- **Edges:** Düğümler arasındaki bağlantılar. Bir sonraki adımı belirlerler. Koşullu olabilirler (örneğin, LLM'in çıktısına göre farklı bir yola gitmek).



Vurgu: Bir `checkpointer` ekleyerek, bu graf her etkileşimden sonra durumunu kalıcı hale getirir ve gerçek bir "hafıza" kazanır.

Zirveye Ulaşmak: Düşünen ve Eyleme Geçen Agent'lar Yaratmak

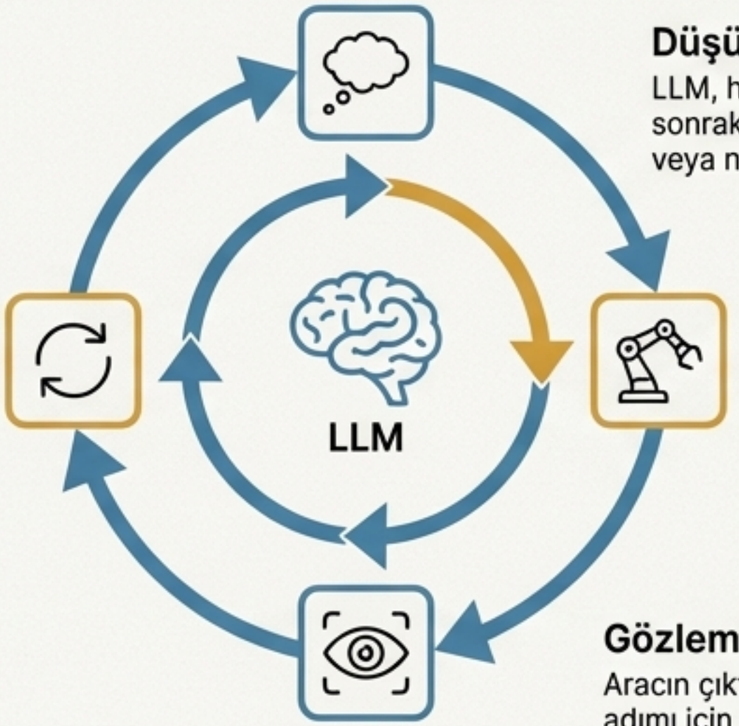
Soru: Uygulamamız artık verilerle konuşabiliyor ve hatırlayabiliyor. Peki ya karmaşık bir görevi başarmak için kendi kendine plan yapabilir, araçları kullanabilir ve kararlar alabilir mi?

Cevap: Agent Mimarileri

Agent, çevresini algılayan, düşünen ve hedeflerine ulaşmak için otonom olarak eyleme geçen bir sistemdir. LangChain/LangGraph, bu tür sistemleri oluşturmak için güçlü bir temel sağlar.

Agent'in Kalbi: ReAct Döngüsü (Reason + Act)

Agent'lar genellikle bir döngü içinde çalışır. Bu döngü, LLM'in kontrolündedir:



Düşün (Reason)
LLM, hedefe ulaşmak için atılması gereken bir sonraki adımı belirler. Bu, bir araç kullanmak veya nihai cevabı vermek olabilir.

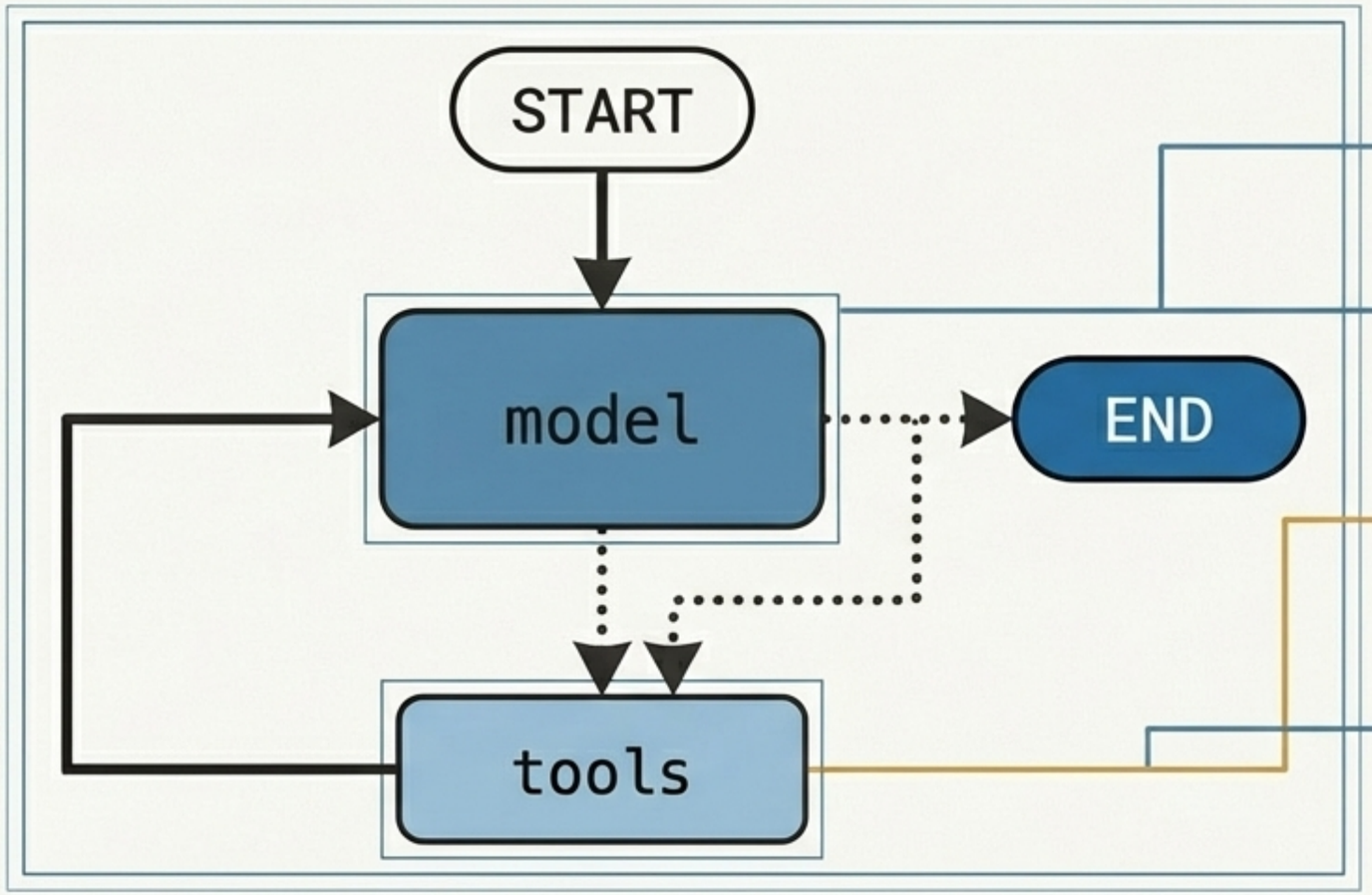
Eyleme Geç (Act)
LLM, bir araç (örneğin, `web_search`, `calculator`) kullanmaya karar verirse, gerekli girdilerle birlikte bu aracı çağırır.

Gözlemle (Observe)
Aracın çıktısı alınır ve bir sonraki düşünme adımı için LLM'e girdi olarak verilir.

Tekrarla
LLM, nihai cevaba ulaştığını düşünene kadar döngü devam eder.

LangGraph ile Bir Agent'ı Hayata Geçirmek

LangGraph, Agent'in durumunu (önceki düşünceler, araç çıktıları) yönetmek ve döngüsel mantığı uygulamak için mükemmel bir araçtır. İşte temel bir Agent grafiği:



Grafiğin Bileşenleri:

State
`messages` listesini içerir. Tüm konuşma geçmişi, düşünceler ve araç çıktıları burada birikir.

Model Düğümü (`model_node`)
Mevcut `messages` listesini alır ve bir sonraki adımı belirler: ya bir `tool_call` ya da son bir cevap içeren bir `AIMessage` döndürür.

Araç Düğümü (`tool_node`)
`model_node` tarafından istenen araç çağrılarını yürütür ve sonuçları bir `ToolMessage` olarak döndürür.

Koşullu Kenar (Conditional Edge)
`model_node`'un çıktısını kontrol eder. Eğer bir `tool_call` varsa, akışı `tool_node`'a yönlendirir. Eğer son cevap varsa, grafiği sonlandırır (END).

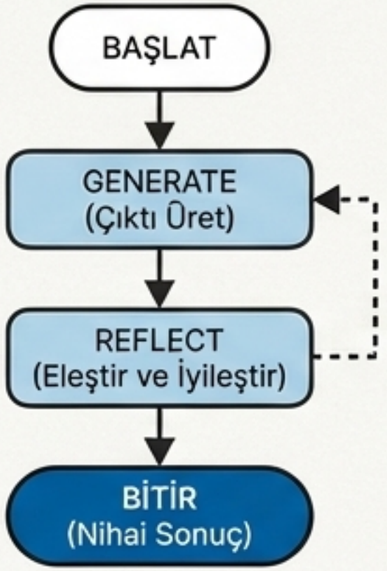
Bu mimari, LLM'e esneklik ve otonomi kazandırır. LLM, görevi tamamlamak için kaç adım atması gerektiğine ve hangi araçları kullanacağına kendisi karar verir.

Agent Yeteneklerini Genişletmek: Gelişmiş Desenler

Temel agent mimarisi güçlü olsa da, karmaşıklık arttıkça performans düşebilir. İşte bu noktada daha gelişmiş mimariler devreye girer.

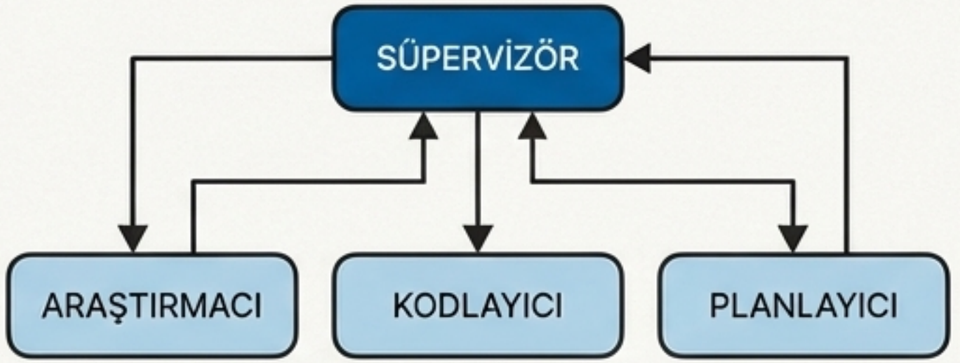
1. Reflection (Öz-Eleştiri)

Agent'ın kendi ürettiği çıktıları (örneğin, bir kod parçası, bir metin taslağı) eleştirmesi ve iyileştirmesi için bir 'eleştirmen' LLM'i döngüye dahil etmek. Bu, 'System 2' düşünmesini taklit ederek daha güvenilir sonuçlar üretir.



2. Multi-Agent Sistemler

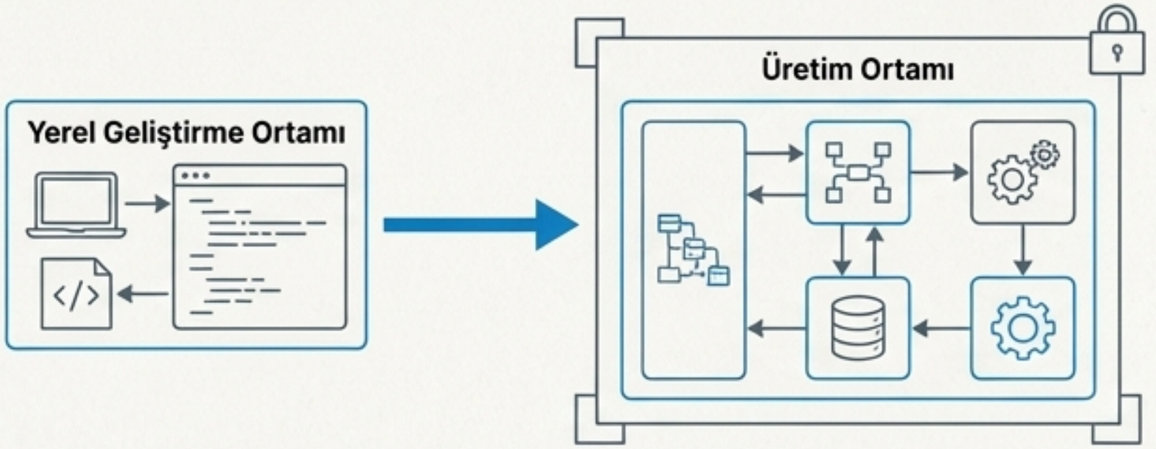
Tek bir büyük agent yerine, her biri belirli bir görevde uzmanlaşmış (örneğin, Araştırmacı, Kodlayıcı, Planlayıcı) birden fazla agent'ı bir araya getirmek. Bir 'Süpervizör' agent, görevi alt görevlere ayırır ve doğru uzmana devreder.



Vurgu: LangGraph'in 'subgraph' özelliği, bu tür modüler ve hiyerarşik multi-agent sistemlerini kolayca oluşturmanızı sağlar.

Son Adım: Agent'ınızı Dünyayla Buluşturmak (Deployment)

Harika bir AI agent'ı geliştirdiniz. Peki şimdi ne olacak? Onu nasıl güvenilir, ölçeklenebilir ve izlenebilir bir şekilde üretime taşıyacaksınız? Bu, yolculuğun en kritik aşamalarından biridir.



Üretim Ortamının Temel Gereksinimleri

Backend API

Backend API ve endpoint söne ot serverter endpoin her gakım ciiyır.

Gözlemleme (Observability)

Gözlemleme, observability semmianan, platformın ɾslandan ve dözzemınır.

Vector Store

Vector Store darbunu strasyagının data ve gönentek verindlir.

Dayanıklılık (Persistence)

Dayanıklılık data stopnat ve yünenine sükeri integresime container.

Çözüm: LangGraph Platform ve LangSmith

LangChain ekosistemi, bu zorlukları aşmak için entegre bir çözüm sunar.

LangGraph Platform → Entegre Çözüm → LangSmith

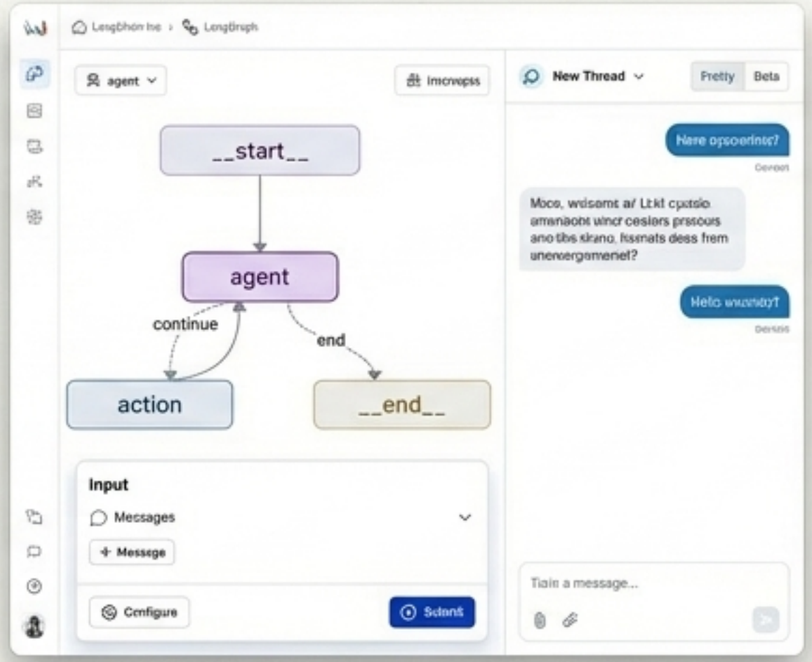
Üretim için Güç Merkezi: LangGraph Platform ve LangSmith

LangGraph Platform: Agent'larınızı Dağıtmak ve Ölçeklendirmek İçin

LangGraph agent'larını büyük ölçekte dağıtmak ve barındırmak için yönetilen bir servistir. Yatay ölçeklendirme, görev kuyrukları ve dayanıklı Postgres checkpointer'ı yöneterek uygulamanızın birçok eşzamanlı kullanıcıyı desteklemesini sağlar.

Source Serif Pro

- Tek tıkla dağıtım (GitHub entegrasyonu)
- Human-in-the-loop kontrolleri
- Double texting yönetimi (Interrupt, Enqueue vb.)
- Asenkron ve zamanlanmış görevler (Cron jobs)



LangSmith: Agent'larınızı Anlamak ve İyileştirmek İçin

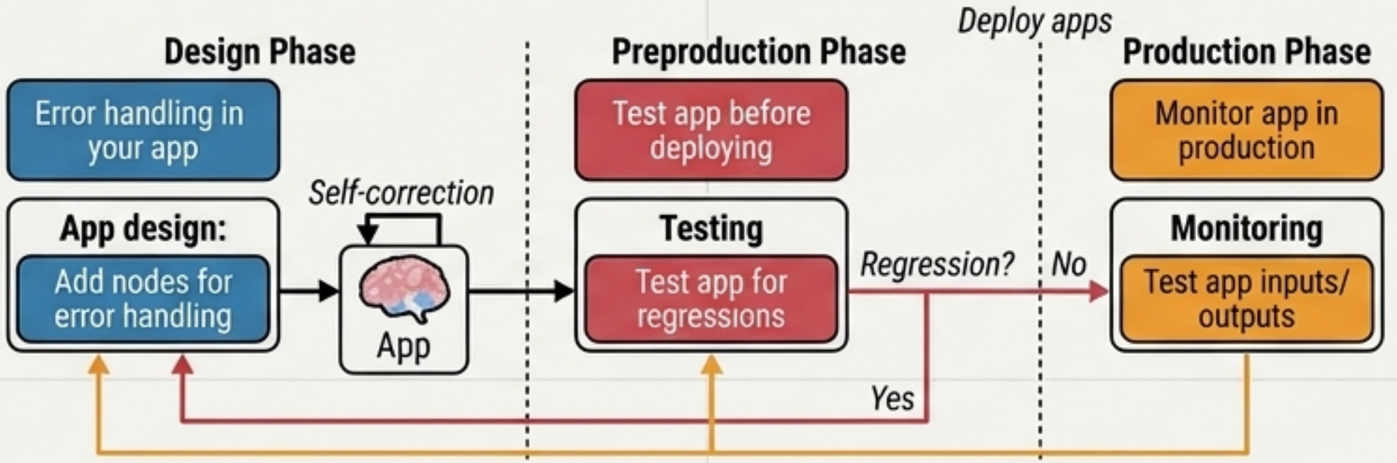
LLM uygulamalarınızı uçtan uca izlemek, hata ayıklamak, test etmek ve değerlendirmek için hepsi bir arada bir geliştirici platformudur.

Source Serif Pro

- **Tracing:** Her bir LLM çağrısını, araç kullanımını ve prompt'u detaylı olarak görme.
- **Evaluation:** Performansı ölçmek için test setleri oluşturma ve otomatik değerlendiriciler (evaluators) çalıştırma.
- **Monitoring:** Üretimdeki uygulamanızın kullanımını, hatalarını, gecikmesini ve maliyetlerini takip etme.
- **Prompt Hub:** Prompt'ları yönetmek ve versiyonlamak.

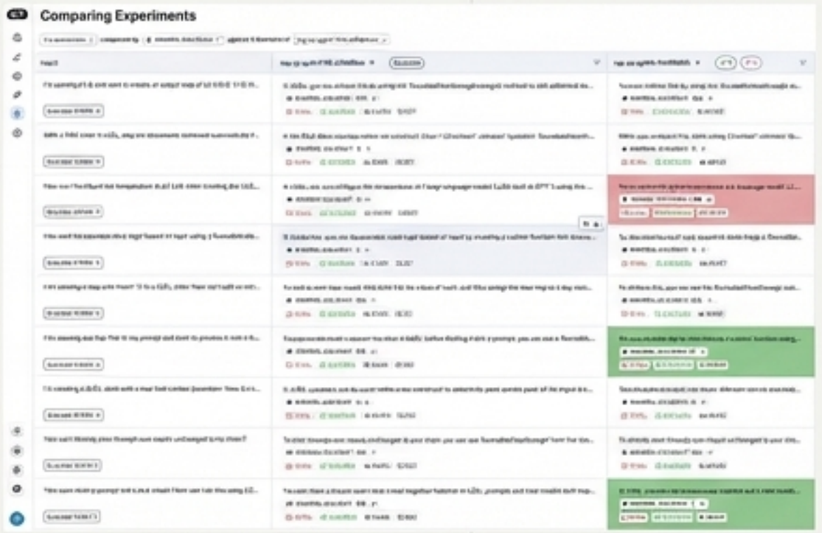
Sürekli İyileştirme Döngüsü: Test ve Değerlendirme

LLM uygulamaları deterministik değildir. Model güncellemeleri veya veri dağılımındaki değişiklikler nedeniyle performans zamanla düşebilir ("model drift"). Bu nedenle, sürekli bir test ve iyileştirme döngüsü kurmak hayati önem taşır.



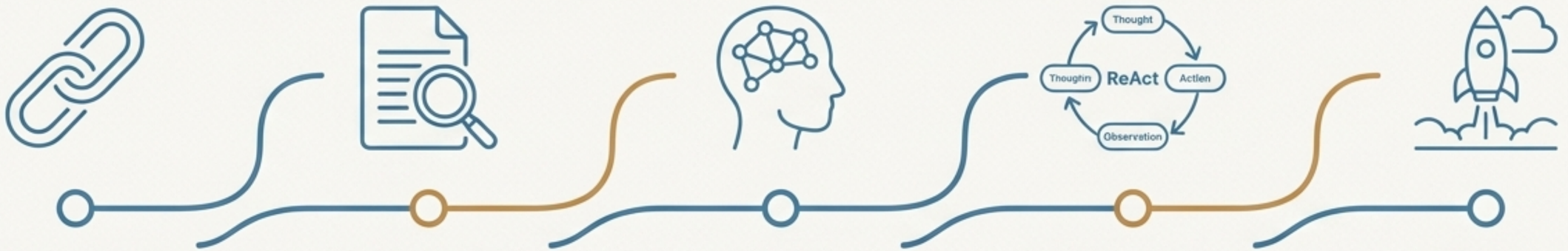
LangSmith ile Değerlendirme Adımları

1. **Dataset Oluşturma**: Uygulamanızı değerlendirmek için girdiler ve beklenen çıktılar içeren örnekler koleksiyonu oluşturun (manuel, loglardan veya sentetik olarak).
2. **Değerlendirici (Evaluator) Tanımlama**:
   - **"Heuristic"**: Çıktının JSON formatında olup olmadığını kontrol etmek gibi kod tabanlı kurallar.
   - **"LLM-as-a-Judge"**: Çıktının doğruluğunu, alaka düzeyini veya diğer niteliksel özelliklerini değerlendirmek için başka bir LLM kullanmak.
   - **"Human Feedback"**: Annotation Queues aracılığıyla insan değerlendirmelerini toplamak.
3. **Regresyon Testi Yapma**: Uygulamanızda yaptığınız bir değişikliğin (örn. yeni bir prompt) performansı düşürüp düşürmediğini anlamak için farklı versiyonları birbiriyle karşılaştırın.



Geliştirici Yolculuğunun Özeti: Yapı Taşlarından Akıllı Sistemlere

Bu sunumda, basit bir LLM çağrısından yola çıkarak, adım adım daha yetenekli ve otonom sistemler inşa etme yolculuğunu tamamladık.



1. Temel Zincir (Chain)

`Prompt -> LLM` - İlk prototip.

2. Veriyle Güçlendirme (RAG)

Kendi verilerinizle konuşma yeteneği.

3. Hafıza (Memory)

`LangGraph` ile durumlu, diyalog kurabilen uygulamalar.

4. Otonomi (Agents)

`ReAct` döngüsü ile düşünen ve eyleme geçen sistemler.

5. Üretim (Production)

`LangSmith` ile dağıtım, izleme ve sürekli iyileştirme.

En güçlü AI uygulamaları, tek bir devasa modelden değil, LangChain gibi araçlarla bir araya getirilmiş, iyi tasarlanmış mimarilerden doğar. Bu sadece bir teknoloji değil, aynı zamanda bir inşa etme sanatı ve bilimidir. **Yolculuğunuz şimdi başlıyor.**