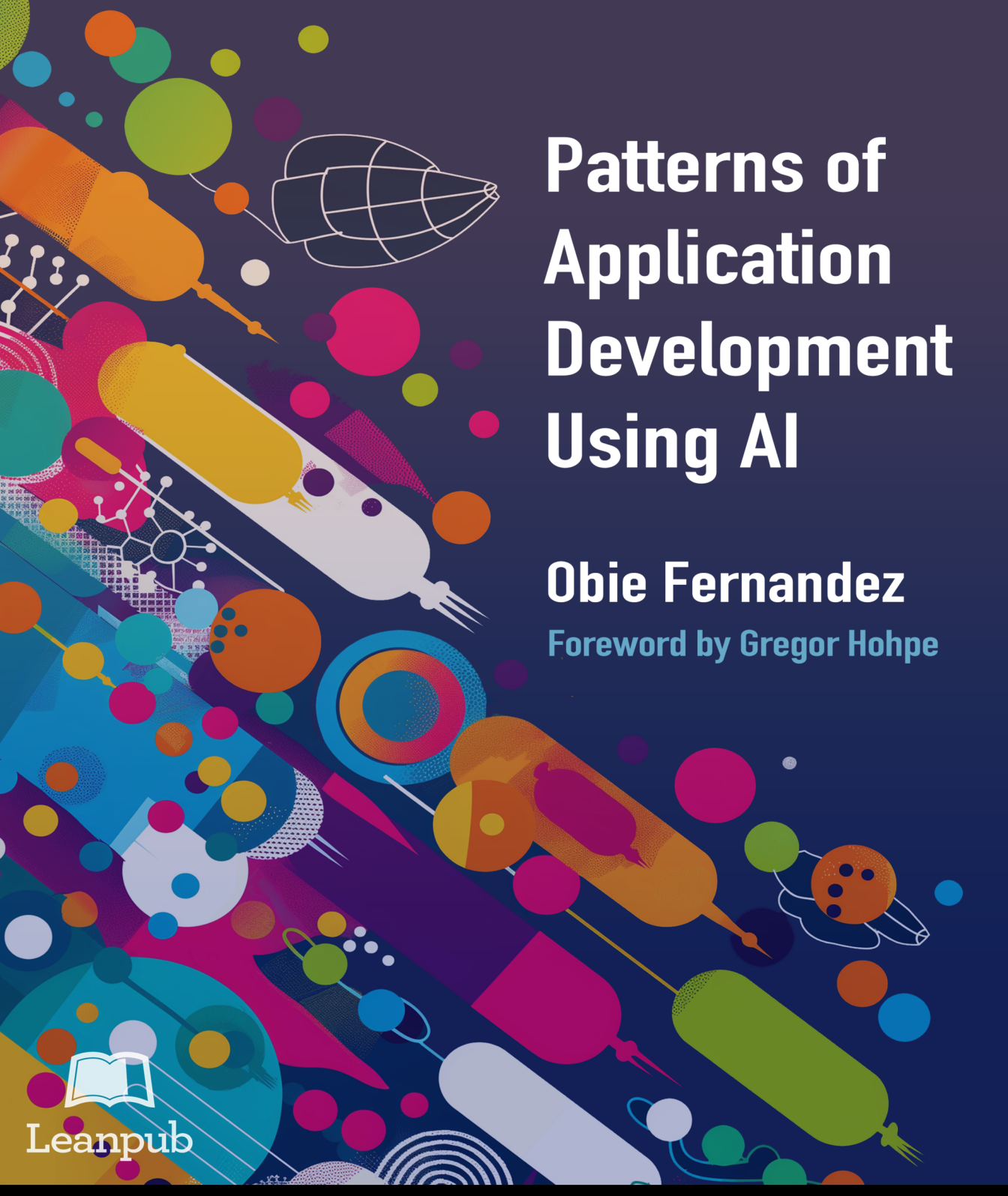


Türkçe Baskı

YZ ile Uygulama Geliştirme Kalıpları (Türkçe Baskı)

Obie Fernandez

Bu kitap http://leanpub.com/patterns-of-application-development-using-ai-tr adresinde satıştadır.

Bu versiyon, 2025-01-23 tarihinde yayınlanmıştır

Bu bir Leanpub kitabıdır. Leanpub, yazar ve yayımcıları Lean Yayımlama sistemi ile destekleyen bir kuruluştur. Lean Yayımlama, henüz çalışma aşamasında olan bir kitabı kullanışlı yollarla destekleyerek, okuyucu geri dönüşünü sağlayan ve prosesi kolaylaştıran bir yöntemdir.

Bu Kitabı Tweetle!

Yazara, Obie Fernandez, destek olmak için bu kitabı Twitter 'da paylaşın!

Bu kitap için önerilen hashtag #poaduai.

Bu linke tıklayarak, bu kitap hakkında Twitter'da neler paylaşıldığını görebilirsiniz:

#poaduai

Muhteşem kraliçem, ilham perim, ışığım ve aşkım Victoria'ya

Ayrıca Obie Fernandez tarafından

Patterns of Application Development Using AI

The Rails 8 Way

The Rails 7 Way

XML The Rails Way

Serverless

El Libro Principiante de Node

The Lean Enterprise

İçindekiler

Kısım 1: Temel Yaklaşımlar ve Teknikler 35

Gregor Hohpe’den Önsöz

Önsöz

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Kitap Hakkında

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Kod Örnekleri Hakkında

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Neleri Kapsamıyorum

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bu Kitap Kimler İçin

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ortak Bir Söz Dağarcığı Oluşturmak

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Katılım

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Teşekkürler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Çizimler neyin nesi?

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

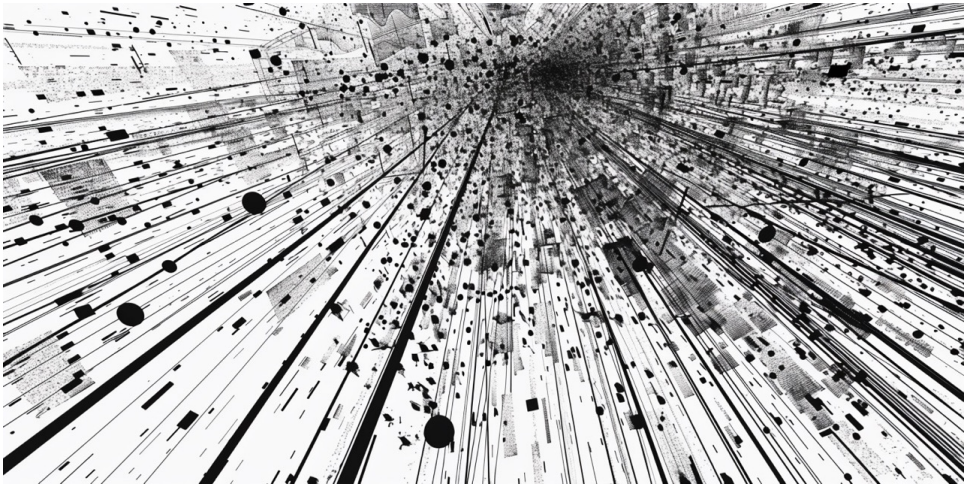
Lean Publishing Hakkında

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Yazar Hakkında

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Giriş



Eğer Yapay Zeka Büyük Dil Modellerini (LLM) programlama projelerinize entegre etmeye hevesliyseniz, ilerleyen bölümlerde sunulan örüntülere ve kod örneklerine hemen dalabilirsiniz. Ancak, bu örüntülerin gücünü ve potansiyelini tam olarak takdir edebilmek için, bunların temsil ettiği daha geniş bağlamı ve bütüncül yaklaşımı anlamak için biraz zaman ayırmaya değer.

Bu örüntüler, yalnızca izole tekniklerin bir koleksiyonu değil, aksine yapay zekayı uygulamalarınıza entegre etmek için birleşik bir çerçevedir. Ben Ruby on Rails kullanıyorum, ancak bu örüntüler hemen hemen diğer tüm programlama ortamlarında da çalışmalıdır. Veri yönetiminden performans optimizasyonuna, kullanıcı deneyiminden güvenliğe kadar geniş bir yelpazedeki konuları ele alarak, geleneksel programlama uygulamalarını yapay zeka yetenekleriyle geliştirmek için kapsamlı bir araç seti sunmaktadır.

Her örüntü kategorisi, uygulamanıza yapay zeka bileşenlerini dahil ederken ortaya çıkan belirli bir zorluğu veya fırsatı ele alır. Bu örüntüler arasındaki ilişkileri ve sinerjileri anlayarak, yapay zekayı nerede ve nasıl en etkili şekilde uygulayacağınız konusunda bilinçli kararlar alabilirsiniz.

Örüntüler asla zorunlu çözümler değildir ve böyle ele alınmamalıdır. Bunlar, kendi benzersiz uygulamanızın özel gereksinimlerine ve kısıtlamalarına göre uyarlanması gereken esnek yapı taşlarıdır. Bu örüntülerin (yazılım alanındaki diğer herhangi bir örüntü gibi) başarılı bir şekilde uygulanması, problem alanının, kullanıcı ihtiyaçlarının ve projenizin genel teknik mimarisinin derinlemesine anlaşılmasına dayanır.

Yazılım Mimarisi Üzerine Düşünceler

1980'lerde programlamaya başladım ve hacker camiasında yer aldım, profesyonel bir yazılım geliştirici olduktan sonra bile hacker zihniyetimi hiç kaybetmedim. En başından beri, fildişi kulelerindeki yazılım mimarlarının gerçekte ne gibi bir değer kattığı konusunda her zaman sağlıklı bir şüphecilik içindeydim.

Bu güçlü yeni yapay zeka teknolojisi dalgasının getirdiği değişimler konusunda kişisel olarak bu kadar heyecanlı olmamın nedenlerinden biri, bunun *yazılım mimarisi* kararları olarak kabul ettiğimiz şeyler üzerindeki etkisidir. Yazılım projelerimizi tasarlama ve uygulama konusundaki "doğru" yolun ne olduğuna dair geleneksel anlayışı sorguluyor. Ayrıca, yapay zeka geliştirmelerinin projenizin herhangi bir bölümünü, herhangi bir zamanda değiştirmeyi her zamankinden daha kolay hale getirmesi nedeniyle, mimarinin hala öncelikle *sistemin değiştirilmesi zor olan kısımları* olarak düşünülüp düşünülemeyeceğini de sorguluyor.

Belki de yazılım mühendisliğinde "post-modern" yaklaşımın zirve yıllarına giriyoruz. Bu bağlamda post-modern, geliştiricilerin her bir kod satırını yazma ve sürdürmeden sorumlu olduğu geleneksel paradigmalardan temel bir kopuşu ifade eder. Bunun yerine, veri manipülasyonu, karmaşık algoritmalar ve hatta uygulama mantığının tüm parçaları

gibi görevleri 3. parti kütüphanelere ve harici API'lere devretme fikrini benimser. Bu post-modern değişim, uygulamaları sıfırdan inşa etme konusundaki geleneksel bilgelikten önemli bir sapmayı temsil eder ve geliştiricileri geliştirme sürecindeki rollerini yeniden düşünmeye zorlar.

Larry Wall ve onun gibi diğer hacker öncülerinin öğretilerinden yola çıkarak, iyi programcıların yalnızca kesinlikle gerekli olan kodu yazdıklarına her zaman inandım. Yazılan kod miktarını en aza indirerek, daha hızlı hareket edebilir, hatalara açık yüzeyi azaltabilir, bakımı basitleştirebilir ve uygulamalarının genel güvenilirliğini artırabiliriz. Daha az kod, diğer işleri başka servislere devrederken temel iş mantığına ve kullanıcı deneyimine odaklanmamıza olanak tanır.

Artık yapay zeka destekli sistemler, daha önce yalnızca insan yazılı kodun alanı olan görevleri halledebildiğine göre, iş değeri ve kullanıcı deneyimi yaratmaya her zamankinden daha fazla odaklanarak daha üretken ve çevik olabilmeliyiz.

Elbette projenizin büyük bölümlerini yapay zeka sistemlerine devretmenin, kontrol kaybı potansiyeli ve sağlam izleme ve geri bildirim mekanizmalarına duyulan ihtiyaç gibi dezavantajları vardır. Bu nedenle, yapay zekanın nasıl çalıştığına dair en azından temel bir anlayış da dahil olmak üzere yeni bir beceri ve bilgi seti gerektirmektedir.

Büyük Dil Modeli Nedir?

Büyük Dil Modelleri (LLM), OpenAI tarafından 2020'de GPT-3'ün piyasaya sürülmesinden bu yana önemli ilgi gören bir yapay zeka modeli türüdür. LLM'ler, insan dilini dikkat çekici bir doğruluk ve akıcılıkla işlemek, anlamak ve üretmek için tasarlanmıştır. Bu bölümde, LLM'lerin nasıl çalıştığına ve neden akıllı sistem bileşenleri oluşturmak için uygun olduklarına kısaca göz atacağız.

Özünde, LLM'ler derin öğrenme algoritmalarına, özellikle sinir ağlarına dayanır. Bu ağlar, bilgiyi işleyen ve ileten birbirine bağlı düğümlerden veya nöronlardan oluşur.

LLM'ler için tercih edilen mimari genellikle metin gibi sıralı verileri işlemede oldukça etkili olduğu kanıtlanan Transformer modelidir.

Dönüştürücü modeller, dikkat mekanizmasına dayanır ve öncelikle doğal dil işleme gibi sıralı veri içeren görevler için kullanılır. Dönüştürücüler, giriş verisini sıralı olarak değil bir bütün olarak işler; bu da uzun mesafeli bağımlılıkları daha etkili bir şekilde yakalamalarını sağlar. Modelin bağlamı ve ilişkileri anlaması için giriş verisinin farklı bölümlerine odaklanmasına yardımcı olan dikkat mekanizması katmanlarına sahiptirler.

BDM'lerin eğitim süreci, modeli kitaplar, makaleler, web siteleri ve kod depoları gibi çok büyük miktarda metinsel veriye maruz bırakmayı içerir. Eğitim sırasında, model metin içindeki kalıpları, ilişkileri ve yapıları tanımayı öğrenir. Dilin dilbilgisi kuralları, kelime ilişkileri ve bağlamsal anlamlar gibi istatistiksel özelliklerini yakalar.

BDM'lerin eğitiminde kullanılan temel tekniklerden biri gözetimsiz öğrenmedir. Bu, modelin açık etiketleme veya rehberlik olmadan veriden öğrendiği anlamına gelir. Eğitim verisindeki kelimelerin ve ifadelerin birlikte bulunma durumlarını analiz ederek kendi başına kalıpları ve temsilleri keşfeder. Bu, BDM'lerin dili ve onun inceliklerini derinden anlamasını sağlar.

BDM'lerin bir diğer önemli yönü *bağlamı* ele alma yetenekleridir. Bir metin parçasını işlerken, BDM'ler sadece tek tek kelimeleri değil, aynı zamanda çevredeki bağlamı da dikkate alır. Metnin anlamını ve amacını anlamak için önceki kelimeleri, cümleleri ve hatta paragrafları göz önünde bulundurur. Bu bağlamsal anlayış, BDM'lerin tutarlı ve alakalı yanıtlar üretmesini sağlar. Belirli bir BDM modelinin yeteneklerini değerlendirmenin ana yollarından biri, yanıt üretmek için dikkate alabildikleri bağlam boyutunu göz önünde bulundurmaktır.

Eğitildikten sonra, BDM'ler çok çeşitli dille ilgili görevler için kullanılabilir. İnsan benzeri metin üretebilir, soruları yanıtlayabilir, belgeleri özetleyebilir, diller arası çeviri yapabilir ve hatta kod yazabilirler. BDM'lerin çok yönlülüğü, kullanıcılarla etkileşime girebilen, metin verilerini işleyip analiz edebilen ve anlamlı çıktılar üretebilen akıllı

sistem bileşenleri oluşturmak için onları değerli kılar.

BDM'leri uygulama mimarisine dahil ederek, kullanıcı girdisini anlayan ve işleyen, dinamik içerik üreten ve akıllı öneriler veya eylemler sunan yapay zeka bileşenleri oluşturabilirsiniz. Ancak BDM'lerle çalışmak, kaynak gereksinimleri ve performans dengelerinin dikkatli bir şekilde değerlendirilmesini gerektirir. BDM'ler hesaplama açısından yoğundur ve çalışmak için önemli miktarda işlem gücü ve bellek (başka bir deyişle, para) gerektirebilir. Çoğumuz, BDM'leri uygulamalarımıza entegre etmenin maliyet etkilerini değerlendirmek ve buna göre hareket etmek zorunda kalacağız.

Çıkarımı Anlamak

Çıkarım, bir modelin yeni, daha önce görmediği verilere dayalı olarak tahminler veya çıktılar ürettiği süreci ifade eder. Bu, eğitilmiş modelin kullanıcı girdilerine yanıt olarak kararlar vermek veya metin, görüntü ya da diğer içerikleri üretmek için kullanıldığı aşamadır.

Eğitim aşamasında, bir yapay zeka modeli, tahminlerindeki hatayı en aza indirmek için parametrelerini ayarlayarak büyük bir veri setinden öğrenir. Eğitildikten sonra, model öğrendiklerini yeni verilere uygulayabilir. Çıkarım, modelin öğrendiği kalıpları ve bilgiyi çıktılar üretmek için kullanma şeklidir.

BDM'ler için çıkarım, bir giriş metni veya istem alıp, *belirteçler* (ki bunlardan yakında bahsedeceğiz) akışı olarak tutarlı ve bağlamsal olarak ilgili bir yanıt üretmeyi içerir. Bu, bir soruyu yanıtlama, bir cümleyi tamamlama, bir hikaye oluşturma veya metin çevirisi yapma gibi birçok görevden biri olabilir.

Sizin ve benim düşünme şeklimizin aksine, bir yapay zeka modelinin çıkarım yoluyla "düşünmesi" tek bir durumsuz işlemde gerçekleşir. Yani, düşünmesi üretim süreciyle sınırlıdır. Size bir soru sorsam ve sadece "bilinç akışı" tarzında bir yanıt kabul etsem gibi, kelimenin tam anlamıyla yüksek sesle düşünmek zorundadır.

Büyük Dil Modelleri Çeşitli Boyut ve Türlerde Gelir

Popüler büyük dil modellerinin (BDM'lerin) neredeyse tümü aynı temel dönüştürücü mimarisi üzerine kurulu ve devasa metin veri setleri üzerinde eğitilmiş olsa da, farklı boyutlarda gelirler ve farklı amaçlar için ince ayar yapılırlar. Bir BDM'nin boyutu, sinir ağındaki parametre sayısıyla ölçülür ve yetenekleri üzerinde büyük bir etkiye sahiptir. 1 ila 2 trilyon parametreye sahip olduğu söylenen GPT-4 gibi daha fazla parametreye sahip büyük modeller, genellikle daha küçük modellerden daha bilgili ve yeteneklidir. Ancak, büyük modeller çalıştırmak için çok daha fazla hesaplama gücü gerektirir ve bu da API çağrıları yoluyla kullandığınızda daha yüksek maliyete dönüşür.

BDM'leri daha pratik ve belirli kullanım durumları için uyarlanmış hale getirmek için, temel modeller genellikle daha hedefli veri setleri üzerinde ince ayar yapılır. Örneğin, bir BDM'ye konuşma yapay zekası için özelleştirmek üzere büyük bir diyalog derlemesi üzerinde eğitim verilebilir. Diğerleri onlara programlama bilgisi kazandırmak için kod üzerinde eğitilir. Hatta kullanıcılarla rol yapma tarzı etkileşimler için özel olarak eğitilmiş modeller bile var!

Geri Getirme ve Üretici Modeller

Büyük dil modelleri (BDM'ler) dünyasında, yanıt üretmek için iki ana yaklaşım vardır: geri getirme tabanlı modeller ve üretici modeller. Her yaklaşımın kendine özgü güçlü ve zayıf yönleri vardır ve aralarındaki farkları anlamak, belirli kullanım senaryonuz için doğru modeli seçmenize yardımcı olabilir.

Geri Getirme Tabanlı Modeller

Geri getirme tabanlı modeller, aynı zamanda bilgi getirme modelleri olarak da bilinir, büyük bir önceden var olan metin veritabanını arayarak ve giriş sorgusuna dayalı olarak

en alakalı pasajları seçerek yanıtlar üretir. Bu modeller yeni metni sıfırdan oluşturmaz, bunun yerine tutarlı bir yanıt oluşturmak için veritabanından alıntıları bir araya getirir.

Geri getirme tabanlı modellerin ana avantajlarından biri, olgusal olarak doğru ve güncel bilgi sağlama yetenekleridir. Düzenlenmiş bir metin veritabanına dayandıkları için, güvenilir kaynaklardan ilgili bilgileri çekebilir ve kullanıcıya sunabilirler. Bu özellik, soru yanıtlama sistemleri veya bilgi tabanları gibi kesin, olgusal cevaplar gerektiren uygulamalar için onları oldukça uygun kılar.

Ancak, geri getirme tabanlı modellerin bazı sınırlamaları vardır. Sadece aradıkları veritabanı kadar iyidirler, bu nedenle veritabanının kalitesi ve kapsamı doğrudan modelin performansını etkiler. Ayrıca, bu modeller veritabanında mevcut metinlerle sınırlı oldukları için tutarlı ve doğal ses çıkaran yanıtlar üretmekte zorlanabilirler.

Bu kitapta saf geri getirme modellerinin kullanımını ele almıyoruz.

Üretici Modeller

Üretici modeller ise, eğitim sırasında öğrendikleri örüntüler ve ilişkilere dayanarak sıfırdan yeni metin oluştururlar. Bu modeller, giriş komutuna uygun yanıtlar üretmek için dil anlayışlarını kullanırlar.

Üretici modellerin ana gücü, yaratıcı, tutarlı ve bağlamsal olarak ilgili metin üretme yetenekleridir. Açık uçlu sohbetlere girebilir, hikayeler oluşturabilir ve hatta kod yazabilirler. Bu özellik onları sohbet robotları, içerik oluşturma ve yaratıcı yazım asistanları gibi daha açık uçlu ve dinamik etkileşimler gerektiren uygulamalar için ideal kılar.

Ancak, üretici modeller bazen tutarsız veya olgusal olarak yanlış bilgiler üretebilirler, çünkü düzenlenmiş bir gerçekler veritabanı yerine eğitim sırasında öğrendikleri örüntülere güvenirler. Ayrıca önyargılara ve halüsinasyonlara daha yatkın olabilirler, makul görünen ancak mutlaka doğru olmayan metinler üretebilirler.

Üretici BDM'lere örnek olarak OpenAI'nin GPT serisi (GPT-3, GPT-4) ve Anthropic'in Claude'u gösterilebilir.

Hibrit Modeller

Ticari olarak mevcut olan birkaç BDM, hem geri getirme hem de üretici yaklaşımları hibrit bir modelde birleştirir. Bu modeller, bir veritabanından ilgili bilgileri bulmak için geri getirme tekniklerini kullanır ve ardından bu bilgileri tutarlı bir yanıta sentezlemek için üretici teknikleri kullanır.

Hibrit modeller, geri getirme tabanlı modellerin olgusal doğruluğunu üretici modellerin doğal dil üretme yetenekleriyle birleştirmeyi amaçlar. Açık uçlu sohbetlere girme yeteneğini korurken daha güvenilir ve güncel bilgiler sağlayabilirler.

Geri getirme tabanlı ve üretici modeller arasında seçim yaparken, uygulamanızın özel gereksinimlerini göz önünde bulundurmalısınız. Eğer temel amaç doğru, olgusal bilgi sağlamaksa, geri getirme tabanlı bir model en iyi seçim olabilir. Uygulama daha açık uçlu ve yaratıcı etkileşimler gerektiriyorsa, üretici bir model daha uygun olabilir. Hibrit modeller iki yaklaşım arasında bir denge sunar ve hem olgusal doğruluk hem de doğal dil üretimi gerektiren uygulamalar için iyi bir seçim olabilir.

Sonuç olarak, geri getirme tabanlı ve üretici modeller arasındaki seçim, belirli kullanım senaryosuna ve doğruluk, yaratıcılık ve esneklik arasındaki ödünleşimlere bağlıdır. Her yaklaşımın güçlü yönlerini ve sınırlamalarını anlayarak bilinçli kararlar verebilirsiniz.

Talimat İnce Ayarı

Talimat ince ayarı, BDM'lerin talimatları daha iyi anlama ve takip etme yeteneğini geliştirmek için insan tarafından yazılmış komutları ve yanıtları kullanan konuşma eğitiminin bir alt kümesidir. İşte insan tarafından yazılmış komutlar ve yanıtlar üzerinde talimat ince ayarının BDM'lerin talimatları anlama ve takip etme yeteneklerini nasıl geliştirdiğini gösteren bazı basit örnekler:

Örnek 1: Müşteri Desteği

Talimat: "İade talep eden bir müşteriye yanıt verin."

İnsan Tarafından Yazılmış Komut: "Son satın aldığım ürün için iade talep etmek istiyorum. Ürün beklentilerimi karşılamadı."

İnsan Tarafından Yazılmış Yanıt: "Ürünün beklentilerinizi karşılamadığını duymak üzüldük. İadenizi işleme almak için lütfen sipariş numaranızı ve iade nedeninizi belirtin. Talebinizi inceleyip 2-3 iş günü içinde size geri dönüş yapacağız."

Örnek 2: Eğitim Desteği

Talimat: "Fotosentez kavramını 10 yaşındaki bir çocuğa açıklayın."

İnsan Tarafından Yazılan İstem: "Bitkiler besinlerini nasıl üretir açıklayabilir misin?"

İnsan Tarafından Yazılan Yanıt: "Tabii ki! Bitkiler fotosentez adı verilen bir süreç ile besinlerini üretirler. Güneş ışığı, su ve karbondioksiti kullanarak besin ve oksijen üretirler. Bitkinin yapraklarında güneş ışığını yakalamalarına yardımcı olan klorofil adında özel bir bölüm vardır."

Örnek 3: Kişisel Asistan

Talimat: "Gelecek Salı saat 15:00'te bir toplantı planla."

> İnsan Tarafından Yazılan İstem: “Salı günü saat 15:00’te bir toplantı ayarlayabilir misin?”
>
> İnsan Tarafından Yazılan Yanıt: “Tabii ki, gelecek Salı saat 15:00 için bir toplantı planladım. Başka bir şeye ihtiyacınız var mı?”

Sonuç olarak, farklı boyutlarda ve farklı uzmanlık alanlarına sahip BDM’lerden oluşan çeşitli bir ekosistem ortaya çıkmıştır. 1-7 milyar parametre aralığındaki daha küçük modeller, daha verimli çalışırken iyi genel dil yetenekleri sunar.

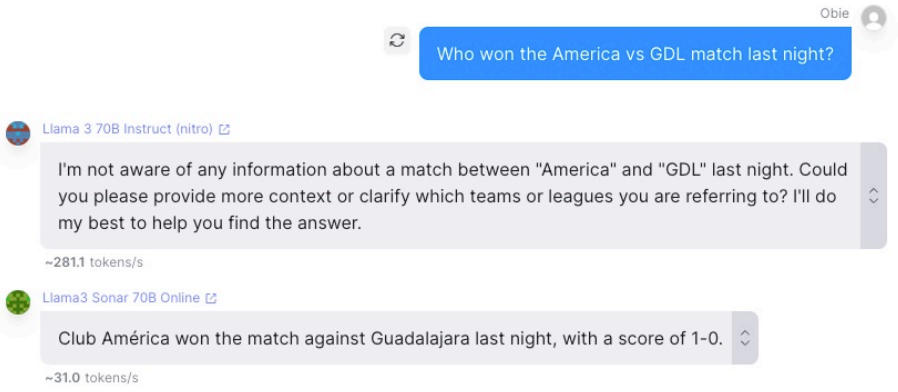
- Mistral 7B
- Llama 3 8B
- Gemma 7B

30-70 milyar parametre civarındaki orta boy modeller, daha güçlü akıl yürütme ve talimat takip etme yetenekleri sunar.

- Llama 3 70B
- Qwen2 70B
- Mixtral 8x22B

Bir uygulamaya BDM entegre ederken, modelin yeteneklerini maliyet, gecikme süresi, bağlam uzunluğu ve içerik filtreleme gibi pratik faktörlerle dengelemeniz gerekir. Daha basit dil görevleri için küçük, talimatlarla ince ayarı yapılmış modeller genellikle en iyi seçimdir; karmaşık akıl yürütme veya analiz için ise en büyük modeller gerekebilir. Modelin eğitim verisi de önemli bir husustur, çünkü modelin bilgi kesme tarihini belirler.

Perplexity gibi bazı modeller gerçek zamanlı bilgi kaynaklarına bağlıdır, bu nedenle etkili bir şekilde kesme tarihleri yoktur. Onlara sorular sorduğunuzda, bağımsız olarak web aramaları yapabilir ve yanıt oluşturmak için rastgele web sayfalarını getirebilirler.



Şekil 1. Çevrimiçi erişimli ve erişimsiz Llama3

Sonuç olarak, her duruma uyan tek bir BDM yoktur. Model boyutu, mimarisi ve eğitimindeki farklılıkları anlamak, belirli bir kullanım durumu için doğru modeli seçmenin anahtarıdır. Farklı modellerle denemeler yapmak, belirli bir görev için en iyi performansı hangilerinin sağladığını ortaya çıkarmanın tek pratik yoludur.

Belirteçleme: Metni Parçalara Ayırma

Büyük bir dil modeli metni işlemeden önce, o metnin *belirteç* adı verilen daha küçük birimlere ayrılması gerekir. Belirteçler tek tek kelimeler, kelimelerin parçaları veya tek karakterler olabilir. Metni belirteçlere ayırma işlemine belirteçleme denir ve bu, veriyi bir dil modeli için hazırlamanın çok önemli bir adımıdır.

Şekil 2. Bu cümle 27 belirteç içerir

Farklı BDM'ler farklı belirteçleme stratejileri kullanır ve bu, modelin performansını ve yeteneklerini önemli ölçüde etkileyebilir. BDM'ler tarafından kullanılan bazı yaygın

belirteçleyiciler şunlardır:

- **GPT (Bayt Çifti Kodlaması)**: GPT belirteçleyicileri, metni alt kelime birimlerine ayırmak için bayt çifti kodlaması (BÇK) adı verilen bir teknik kullanır. BÇK, bir metin dağarcığındaki en sık görülen bayt çiftlerini yinelemeli olarak birleştirerek alt kelime belirteçlerinden oluşan bir sözcük dağarcığı oluşturur. Bu, belirteçleyicinin nadir ve yeni kelimeleri daha yaygın alt kelime parçalarına ayırarak işlemesine olanak tanır. GPT belirteçleyicileri, GPT-3 ve GPT-4 gibi modeller tarafından kullanılır.

- **Llama (SentencePiece)**: Llama belirteçleyicileri, gözetimsiz bir metin belirteçleyici ve belirteç çözücü olan SentencePiece kütüphanesini kullanır. SentencePiece, girdi metnini Unicode karakterlerinin bir dizisi olarak ele alır ve bir eğitim derlemi temelinde alt sözcük dağarcığı öğrenir. Unicode ile kodlanabilen herhangi bir dili işleyebilir, bu da onu çok dilli modeller için oldukça uygun hale getirir. Llama belirteçleyicileri, Meta'nın Llama ve Alpaca gibi modelleri tarafından kullanılır.

- **SentencePiece (Unigram)**: SentencePiece belirteçleyicileri ayrıca Unigram adı verilen, alt sözcük düzenleme tekniğine dayalı farklı bir algoritma da kullanabilir. Unigram belirteçleme, tek tek alt sözcük birimlerine olasılıklar atayan bir unigram dil modeline dayanarak optimal alt sözcük dağarcığını belirler. Bu yaklaşım, BPE'ye kıyasla anlamsal açıdan daha anlamlı alt sözcükler üretebilir. Unigram'lı SentencePiece, Google'ın T5 ve BERT gibi modelleri tarafından kullanılır.

- **Google Gemini (Çok Modlu Belirteçleme)**: Google Gemini, metin, görüntü, ses, video ve kod dahil olmak üzere çeşitli veri türlerini işlemek için tasarlanmış bir belirteçleme şeması kullanır. Bu çok modlu yetenek, Gemini'nin farklı bilgi formlarını işlemesine ve entegre etmesine olanak tanır. Özellikle, Google Gemini 1.5 Pro önceki modellerden çok daha büyük, milyonlarca belirteci işleyebilen bir

bağlam penceresine sahiptir. Bu geniş bağlam penceresi, modelin daha büyük bir bağlamı işlemesini ve potansiyel olarak daha doğru yanıtlar üretmesini sağlar. Ancak, Gemini'nin belirteçleme şemasının diğer modellere göre karakter başına bir belirteç yaklaşımına çok daha yakın olduğunu belirtmek önemlidir. Bu, Google'ın fiyatlandırması belirteçler yerine karakterlere dayalı olduğundan, GPT gibi modelleri kullanmaya alışkınsanız, Gemini modellerini kullanmanın gerçek maliyetinin beklenenden önemli ölçüde daha yüksek olabileceği anlamına gelir.

Belirteçleyici seçimi, bir DDD'nin (Dil Düşünce Modeli) şu yönlerini etkiler:

- **Sözcük dağarcığı boyutu**: Belirteçleyici, modelin tanıdığı benzersiz belirteçler kümesi olan sözcük dağarcığının boyutunu belirler. Daha büyük, daha ayrıntılı bir sözcük dağarcığı, modelin daha geniş bir sözcük ve ifade yelpazesini işlemesine ve hatta çok modlu (metinden fazlasını anlama ve üretme yeteneği) hale gelmesine yardımcı olabilir, ancak bu aynı zamanda modelin bellek gereksinimlerini ve hesaplama karmaşıklığını da artırır.
- **Nadir ve bilinmeyen sözcüklerin işlenmesi**: BPE ve SentencePiece gibi alt sözcük birimleri kullanan belirteçleyiciler, nadir ve bilinmeyen sözcükleri daha yaygın alt sözcük parçalarına ayırabilir. Bu, modelin daha önce görmediği sözcüklerin anlamı hakkında, içerdikleri alt sözcüklere dayanarak eğitimli tahminler yapmasına olanak tanır.
- **Çok dilli destek**: SentencePiece gibi Unicode ile kodlanabilen herhangi bir dili işleyebilen belirteçleyiciler, birden fazla dilde metin işlemesi gereken çok dilli modeller için oldukça uygundur.

Belirli bir uygulama için bir DDD seçerken, kullandığı belirteçleyiciyi ve bunun görevin spesifik dil işleme ihtiyaçlarıyla ne kadar uyumlu olduğunu göz önünde bulundurmak önemlidir. Belirteçleyici, modelin alana özgü terminolojiyi, nadir sözcükleri ve çok dilli metni işleme yeteneği üzerinde önemli bir etkiye sahip olabilir.

Bağlam Boyutu: Bir Dil Modeli Çıkarım Sırasında Ne Kadar Bilgi Kullanabilir?

Dil modellerini tartışırken, bağlam boyutu, bir modelin yanıtlarını işlerken veya üretirken dikkate alabileceği metin miktarını ifade eder. Özünde, modelin "hatırlayabildiği" ve çıktılarını bilgilendirmek için kullanabileceği bilgi miktarının (belirteçler cinsinden ifade edilen) bir ölçüsüdür. Bir dil modelinin bağlam boyutu, yetenekleri ve etkili bir şekilde gerçekleştirebileceği görev türleri üzerinde önemli bir etkiye sahip olabilir.

Bağlam Boyutu Nedir?

Teknik açıdan, bağlam boyutu, bir dil modelinin tek bir girdi dizisinde işleyebileceği belirteç (sözcük veya sözcük parçaları) sayısı ile belirlenir. Bu genellikle modelin "dikkat aralığı" veya "bağlam penceresi" olarak adlandırılır. Bağlam boyutu ne kadar büyükse, model bir yanıt üretirken veya bir görevi gerçekleştirirken bir kerede o kadar çok metni dikkate alabilir.

Farklı dil modellerinin birkaç yüz belirteçten milyonlarca belirtece kadar değişen bağlam boyutları vardır. Referans olarak, tipik bir metin paragrafı yaklaşık 100-150 belirteç içerebilirken, tüm bir kitap on binlerce veya yüz binlerce belirteç içerebilir.

> Transformer tabanlı Büyük Dil Modellerini (DDD) sınırlı bellek ve hesaplama ile sonsuz uzunluktaki girdilere ölçeklendirmek için verimli yöntemler üzerine çalışmalar bile var.

Bağlam Boyutu Neden Önemlidir?

Bir dil modelinin bağlam boyutu, tutarlı ve bağlamsal olarak ilgili metin anlama ve üretme yeteneği üzerinde önemli bir etkiye sahiptir. İşte bağlam boyutunun önemli

olmasının bazı temel nedenleri:

1. **Uzun form içeriği anlama**: Daha büyük bağlam boyutuna sahip modeller, makaleler, raporlar ve hatta tüm kitaplar gibi daha uzun metinleri daha iyi anlayıp analiz edebilir. Bu, belge özetleme, soru cevaplama ve içerik analizi gibi görevler için çok önemlidir.

2. **Tutarlılığı koruma**: Daha geniş bir bağlam penceresi, modelin daha uzun çıktı dizileri boyunca tutarlılık ve devamlılığı korumasına olanak tanır. Bu, tutarlı bir anlatı veya konuyu sürdürmenin önemli olduğu hikaye oluşturma, diyalog sistemleri ve içerik üretimi gibi görevler için önemlidir. Ayrıca BDM'leri yapılandırılmış veri üretmek veya dönüştürmek için kullanırken de kesinlikle çok önemlidir.
3. **Uzun mesafeli bağımlılıkları yakalama**: Bazı dil görevleri, bir metinde birbirinden uzak olan kelimeler veya ifadeler arasındaki ilişkileri anlamayı gerektirir. Daha büyük bağlam boyutuna sahip modeller, duygu analizi, çeviri ve dil anlama gibi görevler için önemli olabilecek bu uzun mesafeli bağımlılıkları yakalamak için daha iyi donanımlıdır.
4. **Karmaşık talimatları işleme**: Dil modellerinin karmaşık, çok adımlı talimatları takip etmek için kullanıldığı uygulamalarda, daha büyük bağlam boyutu, modelin yanıt üretirken sadece en son birkaç kelime yerine talimatların tamamını dikkate almasını sağlar.

Farklı Bağlam Boyutlarına Sahip Dil Modeli Örnekleri

İşte farklı bağlam boyutlarına sahip birkaç dil modeli örneği:

- OpenAI GPT-3.5 Turbo: 4.095 token
- Mistral 7B Instruct: 32.768 token
- Anthropic Claude v1: 100.000 token

- OpenAI GPT-4 Turbo: 128.000 token
- Anthropic Claude v2: 200.000 token
- Google Gemini Pro 1.5: 2,8M token

Gördüğünüz gibi, bu modeller arasında OpenAI GPT-3.5 Turbo modelinin yaklaşık 4.000 tokenından Anthropic Claude v2 modelinin 200.000 tokenına kadar geniş bir bağlam boyutu yelpazesi bulunmaktadır. Google'ın PaLM 2 ve OpenAI'nin GPT-4 gibi bazı modeller, daha uzun giriş dizilerini işleyebilen farklı varyantlar (örneğin "32k" versiyonları) sunmaktadır. Ve şu anda (Nisan 2024) Google Gemini Pro neredeyse 3 milyon token ile övünmektedir!

Bağlam boyutunun belirli bir modelin özel uygulamasına ve sürümüne göre değişebileceğini belirtmekte fayda var. Örneğin, orijinal OpenAI GPT-4 modelinin bağlam boyutu 8.191 token iken, Turbo ve 4o gibi daha sonraki GPT-4 varyantları 128.000 token gibi çok daha büyük bir bağlam boyutuna sahiptir.

> Sam Altman, mevcut bağlam sınırlamalarını 80'lerde kişisel bilgisayar programcılarının uğraşmak zorunda kaldığı kilobaytlık çalışma belleğine benzetmiş ve yakın gelecekte "tüm kişisel verilerinizi" bir büyük dil modelinin bağlamına sığdırabilecek duruma geleceğimizi söylemiştir.

Doğru Bağlam Boyutunu Seçme

Belirli bir uygulama için dil modeli seçerken, söz konusu görevin bağlam boyutu gereksinimlerini dikkate almak önemlidir. Duygu analizi veya basit soru cevaplama gibi kısa, izole metin parçaları içeren görevler için daha küçük bir bağlam boyutu yeterli olabilir. Ancak, daha uzun ve karmaşık metinleri anlama ve üretme gerektiren görevler için daha büyük bir bağlam boyutu muhtemelen gerekli olacaktır.

Daha büyük bağlam boyutlarının genellikle artan hesaplama maliyetleri ve daha yavaş işlem süreleriyle birlikte geldiğini belirtmekte fayda var, çünkü model yanıt üretirken daha fazla bilgiyi dikkate almak zorundadır. Bu nedenle, uygulamanız için bir dil modeli seçerken bağlam boyutu ile performans arasında bir denge kurmanız gerekir.

> Neden en büyük bağlam boyutuna sahip modeli seçip mümkün olduğunca çok bilgiyle doldurmuyoruz? Performans faktörlerinin yanı sıra, diğer ana husus maliyet. Mart 2024'te Google Gemini Pro 1.5 ile tam bağlam kullanarak *tek* bir komut-yanıt döngüsü size neredeyse 8 dolara (USD) mal olacaktır. Bu masrafı haklı çıkaracak bir kullanım senaryonuz varsa, ne âlâ! Ancak çoğu uygulama için, bu kat be kat çok pahalı.

Saman Yığınlarında İğne Bulmak

Büyük veri kümelerinde bilgi erişiminin zorluklarını anlatmak için saman yığınında iğne aramak benzetmesi uzun zamandır kullanılmaktadır. BDM'ler söz konusu olduğunda, bu benzetmeyi biraz değiştiriyoruz. Geniş bir metin içinde (Paul Graham makalelerinin tam bir derlemesi gibi) gömülü tek bir gerçeği değil, metin boyunca dağılmış birden fazla gerçeği aradığımızı düşünün. Bu senaryo, tek bir saman yığınında değil, geniş bir tarlada birden fazla iğne bulmaya benzer. İşte can alıcı nokta: Bu iğneleri sadece bulmakla kalmayıp, onları tutarlı bir şekilde bir araya getirmemiz gerekiyor.

BDM'ler, uzun bağlamlara gömülü birden fazla gerçeği bulup bunlar hakkında akıl yürütmekle görevlendirildiğinde, çifte bir zorlukla karşılaşır. İlk olarak, geri çağırma doğruluğu konusunda basit bir sorun var—gerçek sayısı arttıkça doğal olarak düşüyor. Bu beklenen bir durum; ne de olsa, geniş bir metin boyunca birden fazla ayrıntıyı takip etmek en gelişmiş modelleri bile zorluyor.

İkinci ve belki de daha kritik olan zorluk, bu gerçeklerle akıl yürütme konusudur.

Gerçekleri seçip çıkarmak başka bir şey; onları tutarlı bir anlatıya veya cevaba sentezlemek bambaşka bir şey. Asıl test burada başlıyor. BDM'lerin akıl yürütme görevlerindeki performansı, basit geri çağırma görevlerine kıyasla daha fazla düşme eğiliminde. Bu düşüş sadece hacimle ilgili değil; bağlam, ilgi ve çıkarım arasındaki karmaşık dans ile ilgili.

Peki bu neden oluyor? BDM'lerde bir dereceye kadar yansıtılan insan bilişimindeki bellek ve dikkat dinamiklerini düşünün. Büyük miktarda bilgiyi işlerken BDM'ler, tıpkı insanlar gibi, yeni bilgileri özümserken önceki ayrıntıları kaybedebilir. Bu özellikle, metnin önceki bölümlerini otomatik olarak önceliklendirmek veya yeniden ziyaret etmek üzere açıkça tasarlanmamış modellerde geçerlidir.

Dahası, bir BDM'nin bu geri çağrılan gerçekleri tutarlı bir yanıta dönüştürme yeteneği, anlatı oluşturmaya benzer. Bu sadece bilginin geri çağrılmasını değil, aynı zamanda derin bir anlayış ve bağlamsal yerleştirme gerektirir ki bu da mevcut yapay zeka için zorlu bir görev olmaya devam ediyor.

Peki bu, bu teknolojilerin geliştiricileri ve entegratörleri olarak bizim için ne anlama geliyor? Karmaşık, uzun formlu görevleri ele almak için BDM'lere güvenen sistemler tasarlarken bu sınırlamaların keskin bir şekilde farkında olmalıyız. Belirli koşullar altında performansın düşebileceğini anlamak, gerçekçi beklentiler belirlememize ve daha iyi yedek mekanizmalar veya tamamlayıcı stratejiler geliştirmemize yardımcı olur.

Kipler: Metnin Ötesinde

Günümüzde dil modellerinin çoğu metin işleme ve üretmeye odaklanmış olsa da, görüntüler, ses ve video gibi birden fazla veri türünü doğal olarak girdi ve çıktı olarak alabilen çok kipli modellere doğru artan bir eğilim var. Bu çok kipli modeller, farklı kipler arasında içerik anlayabilen ve üretebilen yapay zeka destekli uygulamalar için yeni olanaklar açıyor.

Kipler Nelerdir?

Dil modelleri bağlamında kipler, bir modelin işleyebileceği ve üretebileceği farklı veri türlerini ifade eder. En yaygın kip, kitaplar, makaleler, web siteleri ve sosyal medya gönderileri gibi çeşitli formlardaki yazılı dili içeren metindir. Ancak, dil modellerine giderek daha fazla dahil edilen başka kipler de vardır:

- **Görüntüler**: Fotoğraflar, illüstrasyonlar ve diyagramlar gibi görsel veriler.
- **Ses**: Konuşma, müzik ve çevresel sesler gibi ses verileri.
- **Video**: Video klipler ve filmler gibi genellikle sesle birlikte gelen hareketli görsel veriler.

Her kip, dil modelleri için benzersiz zorluklar ve fırsatlar sunar. Örneğin, görüntüler modelin görsel kavramları ve ilişkileri anlamasını gerektirirken, ses modelin konuşma ve diğer sesleri işlemesini ve üretmesini gerektirir.

Çok Kipli Dil Modelleri

Çok kipli dil modelleri, tek bir model içinde birden fazla kipi işleyebilecek şekilde tasarlanmıştır. Bu modeller genellikle farklı kiplerde hem girdileri anlayabilen hem de çıktı verisi üretebilen özelleşmiş bileşenlere veya katmanlara sahiptir. Çok kipli dil modellerinin bazı önemli örnekleri şunlardır:

- **OpenAI'nin GPT-4o'su**: GPT-4o, metin yanında konuşma sesini de doğal olarak anlayan ve işleyen büyük bir dil modelidir. Bu yetenek, GPT-4o'nun konuşma dilini yazıya dökme, ses girdilerinden metin üretme ve sözlü sorgulara yanıt verme gibi görevleri gerçekleştirmesine olanak tanır.
- **OpenAI'nin görsel girdili GPT-4'ü**: GPT-4, hem metin hem de görüntüleri işleyebilen büyük bir dil modelidir. Girdi olarak bir görüntü verildiğinde, GPT-4 görüntünün içeriğini analiz edebilir ve görsel bilgileri tanımlayan veya bunlara yanıt veren metin üretebilir.

- **Google'ın Gemini'si**: Gemini, metin, görüntü ve videoyu işleyebilen çok kipli bir modeldir. Görüntü betimleme, video özetleme ve görsel soru yanıtlama gibi görevleri mümkün kılan kipler arası anlama ve üretimi sağlayan birleşik bir mimari kullanır.

- **DALL-E ve Stable Diffusion**: Geleneksel anlamda dil modelleri olmasalar da, bu modeller metin açıklamalarından görüntüler üreterek çoklu modal YZ'nin gücünü göstermektedir. Farklı modaliteler arasında çeviri yapabilen modellerin potansiyelini sergilerler.

Çoklu Modal Modellerin Faydaları ve Uygulamaları

Çoklu modal dil modelleri çeşitli faydalar sunar ve geniş bir uygulama yelpazesini mümkün kılar, bunlar arasında:

- **Gelişmiş anlayış**: Birden fazla modaliteden bilgi işleyerek, bu modeller insanların çeşitli duyusal girdilerden öğrenmesine benzer şekilde dünyayı daha kapsamlı anlayabilirler.
- **Modeller arası üretim**: Çoklu modal modeller bir modaliteden gelen girdiye dayalı olarak başka bir modalitede içerik üretebilir, örneğin metin açıklamasından görüntü oluşturma veya yazılı bir makaleden video özeti oluşturma gibi.
- **Erişilebilirlik**: Çoklu modal modeller, görme engelli kullanıcılar için görüntülerin metin açıklamalarını oluşturma veya yazılı içeriğin sesli versiyonlarını oluşturma gibi modaliteler arası çeviri yaparak bilgiyi daha erişilebilir hale getirebilir.
- **Yaratıcı uygulamalar**: Çoklu modal modeller, metin komutlarına dayalı sanat, müzik veya video oluşturma gibi yaratıcı görevler için kullanılabilir ve sanatçılar ile içerik üreticileri için yeni olanaklar sunar.

Çoklu modal dil modelleri geliştikçe, birden fazla modalitede içerik anlayabilen ve üretebilen YZ destekli uygulamaların geliştirilmesinde giderek daha önemli bir rol

oynayacakları muhtemeldir. Bu, insanlar ve YZ sistemleri arasında daha doğal ve sezgisel etkileşimleri mümkün kılacak ve yaratıcı ifade ile bilgi yayılımı için yeni olanakların önünü açacaktır.

Sağlayıcı Ekosistemleri

Büyük dil modellerini (LLM'leri) uygulamalara dahil etmek söz konusu olduğunda, seçebileceğiniz giderek artan bir seçenek yelpazesi bulunmaktadır. OpenAI, Anthropic, Google ve Cohere gibi her büyük LLM sağlayıcısı, kendi model, API ve araç ekosistemini sunar. Doğru sağlayıcıyı seçmek, fiyatlandırma, performans, içerik filtreleme, veri gizliliği ve özelleştirme seçenekleri gibi çeşitli faktörleri göz önünde bulundurmayı gerektirir.

OpenAI

OpenAI, GPT serisi (GPT-3, GPT-4) çeşitli uygulamalarda yaygın olarak kullanılan en tanınmış LLM sağlayıcılarından biridir. OpenAI, modellerini uygulamalara kolayca entegre etmenizi sağlayan kullanıcı dostu bir API sunar. Giriş seviyesi Ada modelinden güçlü Davinci modeline kadar farklı yeteneklere ve fiyat noktalarına sahip bir dizi model sunarlar.

OpenAI'nin ekosistemi ayrıca, komutlarla deney yapmanıza ve modelleri belirli kullanım durumları için ince ayar yapmanıza olanak tanıyan OpenAI Playground gibi araçları da içerir. Uygunsuz veya zararlı içerik üretimini önlemeye yardımcı olan içerik filtreleme seçenekleri sunarlar.

OpenAI'nin modellerini doğrudan kullanırken, Alex Rudall'ın ruby-openai kütüphanesine güveniyorum.

Anthropic

Anthropic, Claude modellerinin güçlü performans ve etik değerlendirmeler açısından popülerlik kazandığı LLM alanındaki bir diğer önemli oyuncudur. Anthropic, içerik

filtreleme ve zararlı çıktılardan kaçınma konusunda güçlü bir vurguyla güvenli ve sorumlu YZ sistemleri geliştirmeye odaklanır.

Anthropic'in ekosistemi, modeli uygulamalarına entegre etmenizi sağlayan Claude API'sinin yanı sıra komut mühendisliği ve ince ayar için araçları da içerir. Ayrıca, daha güncel ve gerçeğe dayalı yanıtlar için web arama yeteneklerini içeren Claude Instant modelini de sunarlar.

Anthropic'in modellerini doğrudan kullanırken, Alex Rudall'ın anthrophic kütüphanesine güveniyorum.

Google

Google, Gemini, BERT, T5 ve PaLM dahil olmak üzere birkaç güçlü LLM geliştirmiştir. Bu modeller, çok çeşitli doğal dil işleme görevlerindeki güçlü performanslarıyla tanınır. Google'ın ekosistemi, makine öğrenimi modellerini oluşturmak ve eğitmek için araçlar ve çerçeveler sağlayan TensorFlow ve Keras kütüphanelerini içerir.

Google ayrıca, modellerini bulutta kolayca dağıtmanıza ve ölçeklendirmenize olanak tanıyan bir Cloud AI Platform sunar. Duygu analizi, varlık tanıma ve çeviri gibi görevler için bir dizi önceden eğitilmiş model ve API sağlarlar.

Meta

Meta, eski adıyla Facebook, LLaMA ve OPT gibi modellerin yayınlanmasıyla öne çıkan büyük dil modellerinin geliştirilmesine derinden yatırım yapmıştır. Bu modeller, çeşitli dil görevlerindeki güçlü performanslarıyla öne çıkar ve Meta'nın araştırma ve topluluk iş birliğine olan bağlılığını destekleyerek büyük ölçüde açık kaynak kanalları aracılığıyla kullanıma sunulur.

Meta'nın ekosistemi öncelikle, dinamik hesaplama yetenekleri ve esnekliği ile yenilikçi YZ araştırma ve geliştirmesini kolaylaştıran, açık kaynaklı bir makine öğrenimi kütüphanesi olan PyTorch etrafında inşa edilmiştir.

Meta, teknik hizmetlerinin yanı sıra etik yapay zeka geliştirmeye büyük önem vermektedir. Güçlü içerik filtreleme uygular ve yapay zeka uygulamalarında güvenlik ve sorumluluk şeklindeki daha geniş hedefleriyle uyumlu olarak önyargıları azaltmaya odaklanır.

Cohere

Cohere, BDM alanına daha yeni giren ve BDM'leri rakiplerinden daha erişilebilir ve kullanımı kolay hale getirmeye odaklanan bir şirkettir. Ekosistemleri, metin oluşturma, sınıflandırma ve özetleme gibi görevler için önceden eğitilmiş modellere erişim sağlayan Cohere API'sini içerir.

Cohere ayrıca bildirim mühendisliği, ince ayar ve içerik filtreleme için araçlar sunar. Şifrelenmiş veri depolama ve erişim kontrolleri gibi özelliklerle veri gizliliği ve güvenliğine önem verirler.

Ollama

Ollama, kullanıcıların çeşitli büyük dil modellerini (BDM'ler) kendi makinelerinde yerel olarak yönetmelerine ve dağıtmalarına olanak tanıyan, harici bulut hizmetlerine güvenmeden AI modellerinin tam kontrolünü sağlayan kendi kendine barındırılan bir platformdur. Bu kurulum, veri gizliliğine öncelik veren ve AI operasyonlarını kurum içinde yönetmek isteyenler için idealdir.

Platform, boyut ve hesaplama gereksinimleri açısından farklılık gösteren Llama, Phi, Gemma ve Mistral sürümlerini içeren çeşitli modelleri destekler. Ollama, bu modelleri `ollama run <model_name>` gibi basit komutlar kullanarak doğrudan komut satırından indirmeyi ve çalıştırmayı kolaylaştırır ve macOS, Linux ve Windows dahil olmak üzere farklı işletim sistemlerinde çalışacak şekilde tasarlanmıştır.

Uzak bir API kullanmadan açık kaynaklı modelleri uygulamalarına entegre etmek isteyen geliştiriciler için Ollama, konteyner yönetim araçlarına benzer şekilde model

yaşam döngülerini yönetmek için bir CLI sunar. Ayrıca, modelleri belirli ihtiyaçlara veya kullanım durumlarına göre uyarlamak için yüksek düzeyde özelleştirme sağlayan özel yapılandırmaları ve bildirimleri destekler.

Ollama, komut satırı arayüzü ve AI modellerinin yönetiminde ve dağıtımında sunduğu esneklik nedeniyle özellikle teknoloji konusunda bilgili kullanıcılar ve geliştiriciler için uygundur. Bu, güvenlik ve kontrolden ödün vermeden güçlü AI yeteneklerine ihtiyaç duyan işletmeler ve bireyler için güçlü bir araç haline getirir.

Çoklu Model Platformları

Ek olarak, Together.ai ve Groq gibi çok çeşitli açık kaynaklı modelleri barındıran sağlayıcılar da vardır. Bu platformlar esneklik ve özelleştirme sunar, açık kaynaklı modelleri çalıştırmanıza ve bazı durumlarda özel ihtiyaçlarınıza göre ince ayar yapmanıza olanak tanır. Örneğin, Together.ai, kullanıcıların farklı modellerle ve yapılandırmalarla deney yapmasını sağlayan çeşitli açık kaynaklı BDM'lere erişim sağlar. Groq, bu kitabın yazıldığı tarihte neredeyse sihirli görünen ultra yüksek performanslı tamamlamaya odaklanır.

BDM Sağlayıcısı Seçimi

Bir BDM sağlayıcısı seçerken şu faktörleri göz önünde bulundurmalısınız:

- **Fiyatlandırma**: Farklı sağlayıcılar, kullandıkça öde modelinden abonelik tabanlı planlara kadar değişen fiyatlandırma modelleri sunar. Bir sağlayıcı seçerken beklenen kullanımı ve bütçeyi göz önünde bulundurmak önemlidir.
- **Performans**: BDM'lerin performansı sağlayıcılar arasında önemli ölçüde farklılık gösterebilir, bu nedenle karar vermeden önce modelleri belirli kullanım durumlarında karşılaştırmak ve test etmek önemlidir.
- **İçerik Filtreleme**: Uygulamaya bağlı olarak, içerik filtreleme kritik bir husus olabilir. Bazı sağlayıcılar diğerlerinden daha güçlü içerik filtreleme seçenekleri sunar.

- **Veri Gizliliği**: Uygulama hassas kullanıcı verilerini işliyorsa, güçlü veri gizliliği ve güvenlik uygulamalarına sahip bir sağlayıcı seçmek önemlidir.
- **Özelleştirme**: Bazı sağlayıcılar, belirli kullanım durumları için modelleri ince ayarlama ve özelleştirme konusunda daha fazla esneklik sunar.

Sonuç olarak, BDM sağlayıcısı seçimi, uygulamanın özel gereksinimlerine ve kısıtlamalarına bağlıdır. Fiyatlandırma, performans ve veri gizliliği gibi faktörleri dikkatlice değerlendirerek, ihtiyaçlarınızı en iyi karşılayan sağlayıcıyı seçebilirsiniz.

BDM alanının sürekli geliştiğini ve düzenli olarak yeni sağlayıcıların ve modellerin ortaya çıktığını da belirtmekte fayda var. En son gelişmelerden haberdar olmalı ve yeni seçenekleri keşfetmeye açık olmalısınız.

OpenRouter

Bu kitap boyunca API sağlayıcısı olarak yalnızca OpenRouter'ı kullanacağım. Bunun nedeni basit: en popüler ticari ve açık kaynaklı modeller için tek bir durak noktası olması. Eğer biraz AI kodlaması yapmak için sabırsızlanıyorsanız, başlamak için en iyi yerlerden biri benim OpenRouter Ruby Kütüphanem.

Performans Üzerine Düşünmek

Dil modellerini uygulamalara entegre ederken, performans kritik bir değerlendirme faktörüdür. Bir dil modelinin performansı, *gecikme süresi* (yanıt üretmek için gereken süre) ve *verim* (birim zamanda işleyebildiği istek sayısı) açısından ölçülebilir.

İlk Simgeye Kadar Geçen Süre (TTFT), özellikle sohbet robotları ve etkileşimli, gerçek zamanlı yanıtlar gerektiren uygulamalar için bir diğer önemli performans metriğidir. TTFT, kullanıcının isteğinin alınmasından yanıtın ilk kelimesinin (veya simgesinin) üretilmesine kadar geçen gecikme süresini ölçer. Bu metrik, kesintisiz ve etkileyici

bir kullanıcı deneyiminin sürdürülmesi için çok önemlidir, çünkü geciken yanıtlar kullanıcının hayal kırıklığına uğramasına ve ilgisinin dağılmasına yol açabilir.

Bu performans metrikleri, kullanıcı deneyimi ve uygulamanın ölçeklenebilirliği üzerinde önemli bir etkiye sahip olabilir.

Bir dil modelinin performansını etkileyen çeşitli faktörler vardır:

Parametre Sayısı: Daha fazla parametreye sahip büyük modeller, genellikle daha fazla hesaplama kaynağı gerektirir ve küçük modellere kıyasla daha yüksek gecikme süresine ve daha düşük verime sahip olabilir.

Donanım: Bir dil modelinin performansı, üzerinde çalıştığı donanıma bağlı olarak önemli ölçüde değişebilir. Bulut sağlayıcıları, makine öğrenimi iş yükleri için optimize edilmiş GPU ve TPU örnekleri sunar ve bunlar model çıkarımını büyük ölçüde hızlandırabilir.

OpenRouter'ın güzel yanlarından biri, sunduğu modellerin çoğu için farklı performans profilleri ve maliyetlere sahip çeşitli bulut sağlayıcıları arasından seçim yapabilmenizdir.

Nicemleme: Nicemleme teknikleri, ağırlıkları ve aktivasyonları daha düşük hassasiyetli veri türleriyle temsil ederek bir modelin bellek ayak izini ve hesaplama gereksinimlerini azaltmak için kullanılabilir. Bu, kaliteden önemli ölçüde ödün vermeden performansı artırabilir. Bir uygulama geliştiricisi olarak muhtemelen farklı nicemleme seviyelerinde kendi modellerinizi eğitmekle uğraşmayacaksınız, ancak en azından terminolojiye aşina olmak faydalıdır.

Toplu İşleme: Birden fazla isteği eş zamanlı olarak toplu halde işlemek, model yükleme ve veri aktarımı yükünü azaltarak verimi artırabilir.

Önbellekleme: Sık kullanılan istemler veya girdi dizileri için sonuçların önbelleğe alınması, çıkarım isteklerinin sayısını azaltabilir ve genel performansı iyileştirebilir.

Üretim ortamında kullanılacak bir dil modeli seçerken, temsili iş yükleri ve donanım yapılandırmaları üzerinde performans karşılaştırması yapmak önemlidir. Bu, potansiyel darboğazları belirlemeye ve modelin gerekli performans hedeflerini karşılayabileceğinden emin olmaya yardımcı olabilir.

Model performansı ile maliyet, esneklik ve entegrasyon kolaylığı gibi diğer faktörler arasındaki dengeyi de göz önünde bulundurmak önemlidir. Örneğin, gerçek zamanlı yanıt gerektiren uygulamalar için daha düşük gecikme süresine sahip, daha küçük ve daha ekonomik bir model tercih edilebilirken, toplu işleme veya karmaşık akıl yürütme görevleri için daha büyük ve güçlü bir model daha uygun olabilir.

Farklı DDD Modelleriyle Deney Yapmak

Bir DDD'yi seçmek nadiren kalıcı bir karardır. Düzenli olarak yeni ve geliştirilmiş modeller yayınlandığından, uygulamaları zaman içinde farklı dil modellerinin değiştirilebilmesine olanak tanıyan modüler bir şekilde oluşturmak iyidir. İstemler ve veri setleri genellikle minimal değişikliklerle modeller arasında yeniden kullanılabilir. Bu, uygulamalarınızı tamamen yeniden tasarlamak zorunda kalmadan dil modellemedeki en son gelişmelerden yararlanmanızı sağlar.

Çok çeşitli model seçenekleri arasında kolayca geçiş yapabilme yeteneği, OpenRouter'ı sevmemin bir başka nedenidir.

Yeni bir dil modeline geçerken, uygulamanın gereksinimlerini karşıladığından emin olmak için performansını ve çıktı kalitesini kapsamlı bir şekilde test etmek ve doğrulamak önemlidir. Bu, modelin alana özgü veriler üzerinde yeniden eğitilmesini veya ince ayar yapılmasını ve modelin çıktılarına bağlı olan downstream bileşenlerin güncellenmesini içerebilir.

Uygulamaları performans ve modülerlik göz önünde bulundurularak tasarlayarak, hızla gelişen dil modelleme teknolojisine uyum sağlayabilen ölçeklenebilir, verimli ve

geleceğe dönük sistemler oluşturabilirsiniz.

Bileşik Yapay Zeka Sistemleri

Giriş bölümümüzü kapatmadan önce, ChatGPT ile tetiklenen üretici yapay zeka alanındaki ilgi patlamasından önceki 2023 yılına kadar, geleneksel yapay zeka yaklaşımlarının genellikle tek, kapalı modellerin entegrasyonuna dayandığını belirtmekte fayda var. Buna karşılık, *Bileşik Yapay Zeka Sistemleri* akıllı davranışı elde etmek için birlikte çalışan birbirine bağlı bileşenlerin karmaşık süreçlerinden yararlanır.

Özünde, bileşik yapay zeka sistemleri, her biri belirli görevleri veya işlevleri yerine getirmek üzere tasarlanmış çoklu modüllerden oluşur. Bu modüller üreticileri, getirici sistemleri, sıralayıcıları, sınıflandırıcıları ve çeşitli diğer özelleşmiş bileşenleri içerebilir. Genel sistemi daha küçük, odaklanmış birimlere ayırarak, geliştiriciler daha esnek, ölçeklenebilir ve sürdürülebilir yapay zeka mimarileri oluşturabilir.

Bileşik yapay zeka sistemlerinin temel avantajlarından biri, farklı yapay zeka tekniklerini ve modellerini birleştirme yetenekleridir. Örneğin, bir sistem doğal dil anlama ve üretimi için büyük dil modeli (LLM) kullanırken, bilgi getirme veya kural tabanlı karar verme için ayrı bir model kullanabilir. Bu modüler yaklaşım, tek tip bir çözüme güvenmek yerine, her özel görev için en iyi araç ve teknikleri seçmenize olanak tanır.

Ancak, bileşik yapay zeka sistemleri oluşturmak kendine özgü zorluklar da sunar. Özellikle, sistemin davranışının genel tutarlılığını ve uyumluluğunu sağlamak, sağlam test, izleme ve yönetişim mekanizmaları gerektirir.

GPT-4 gibi güçlü LLM'lerin ortaya çıkışı, bu gelişmiş modellerin doğal dil anlama yeteneklerinin yanı sıra sınıflandırma, sıralama ve üretim gibi bileşik sistem içindeki birden çok rolü üstlenebilmeleri sayesinde, bileşik yapay zeka sistemleriyle her zamankinden daha kolay deney yapmamızı sağlıyor. Bu çok yönlülük, geliştiricilerin bileşik yapay zeka mimarilerini hızlıca prototiplemesine ve yinelemesine olanak tanıyarak, akıllı uygulama geliştirme için yeni olanaklar açıyor.

Bileşik Yapay Zeka Sistemleri için Dağıtım Desenleri

Bileşik yapay zeka sistemleri, belirli gereksinimleri ve kullanım senaryolarını ele almak üzere tasarlanmış çeşitli desenler kullanılarak dağıtılabilir. Dört yaygın dağıtım desenini inceleyelim: Soru ve Cevap, Çoklu Ajan/Ajanlı Problem Çözücüler, Konuşma Yapay Zekası ve CoPilot'lar.

Soru ve Cevap

Soru ve Cevap (S&C) sistemleri, basit bir arama motorundan daha fazlası olarak işlev görmek için yapay zeka modellerinin anlama yetenekleriyle geliştirilmiş bilgi getirmeye odaklanır. Güçlü dil modellerini Geri Getirme Destekli Üretim (RAG) kullanarak harici bilgi kaynaklarıyla birleştirerek, Soru ve Cevap sistemleri halüsinasyonlardan kaçınır ve kullanıcı sorgularına doğru ve bağlamsal olarak ilgili yanıtlar sağlar.

LLM tabanlı bir S&C sisteminin temel bileşenleri şunları içerir:

- **Sorgu anlama ve yeniden formüle etme**: Kullanıcı sorgularını analiz etme ve altta yatan bilgi kaynaklarıyla daha iyi eşleşecek şekilde yeniden formüle etme.
- **Bilgi getirme**: Yeniden formüle edilmiş sorguya dayalı olarak yapılandırılmış veya yapılandırılmamış veri kaynaklarından ilgili bilgileri getirme.
- **Yanıt üretme**: Getirilen bilgiyi dil modelinin üretim yetenekleriyle bütünleştirerek tutarlı ve bilgilendirici yanıtlar oluşturma.

RAG alt sistemleri özellikle müşteri desteği, bilgi yönetimi veya eğitim uygulamaları gibi doğru ve güncel bilgi sağlamanın kritik olduğu S&C alanlarında önemlidir.

Çoklu Ajan/Ajanlı Problem Çözücüler

Ajanlı olarak da bilinen çoklu ajan sistemleri, karmaşık problemleri çözmek için birlikte çalışan birden fazla otonom ajandan oluşur. Her ajanın belirli bir rolü, beceri seti ve ilgili araçlara veya bilgi kaynaklarına erişimi vardır. Bu ajanlar işbirliği yaparak ve bilgi alışverişinde bulunarak, tek bir ajanın tek başına ele almasının zor veya imkansız olacağı görevleri çözebilirler.

Çoklu ajan problem çözücülerin temel ilkeleri şunları içerir:

- **Uzmanlaşma**: Her ajan, kendine özgü yeteneklerini ve bilgisini kullanarak problemin belirli bir yönüne odaklanır.
- **İşbirliği**: Ajanlar genellikle mesaj iletimi veya paylaşılan bellek yoluyla ortak bir hedefe ulaşmak için iletişim kurar ve eylemlerini koordine eder.
- **Uyarlanabilirlik**: Sistem, tek tek ajanların rollerini ve davranışlarını ayarlayarak değişen koşullara veya gereksinimlere uyum sağlayabilir.

Çoklu ajan sistemleri, tedarik zinciri optimizasyonu, trafik yönetimi veya acil durum müdahale planlaması gibi dağıtık problem çözme gerektiren uygulamalar için uygundur.

Konuşma Yapay Zekası

Konuşma yapay zekası sistemleri, kullanıcılar ile akıllı ajanlar arasında doğal dil etkileşimlerini mümkün kılar. Bu sistemler, ilgi çekici ve kişiselleştirilmiş konuşma deneyimleri sağlamak için doğal dil anlama, diyalog yönetimi ve dil üretimi yeteneklerini birleştirir.

Bir konuşma yapay zekası sisteminin ana bileşenleri şunları içerir:

- **Niyet tanıma**: Soru sorma, istekte bulunma veya duygu ifade etme gibi kullanıcının girdisine dayalı niyetini tanımlama.
- **Varlık çıkarımı**: Kullanıcının girdisinden tarihler, konumlar veya ürün adları gibi ilgili varlıkları veya parametreleri çıkarma.
- **Diyalog yönetimi**: Konuşmanın durumunu sürdürme, kullanıcının niyeti ve bağlama dayalı olarak uygun yanıtı belirleme ve çok turlu etkileşimleri yönetme.
- **Yanıt üretme**: Dil modelleri, şablonlar veya geri getirme tabanlı yöntemler kullanarak insana benzer yanıtlar üretme.

Konuşma yapay zekası sistemleri yaygın olarak müşteri hizmetleri sohbet robotlarında, sanal asistanlarda ve sesle kontrol edilen arayüzlerde kullanılır. Daha önce belirtildiği gibi, bu kitaptaki yaklaşımların, desenlerin ve kod örneklerinin çoğu, Olympia adlı büyük bir konuşma yapay zekası sistemi üzerindeki çalışmalarımdan doğrudan alınmıştır.

CoPilot'lar

CoPilot'lar, insan kullanıcıların yanında çalışarak onların üretkenliğini ve karar verme yeteneklerini artıran YZ destekli asistanlardır. Bu sistemler, akıllı öneriler sunmak, görevleri otomatikleştirmek ve bağlamsal destek sağlamak için doğal dil işleme, makine öğrenimi ve alana özgü bilgilerin bir kombinasyonunu kullanır.

CoPilot'ların temel özellikleri şunlardır:

- **Kişiselleştirme**: Bireysel kullanıcı tercihlerine, iş akışlarına ve iletişim stillerine uyum sağlama.
- **Proaktif yardım**: Kullanıcı ihtiyaçlarını öngörme ve açık komutlar olmadan ilgili öneriler veya eylemler sunma.
- **Sürekli öğrenme**: Kullanıcı geri bildirimleri, etkileşimler ve verilerden öğrenerek zaman içinde performansını iyileştirme.

CoPilot'lar, yazılım geliştirme (örn. kod tamamlama ve hata tespiti), yaratıcı yazarlık (örn. içerik önerileri ve düzenleme) ve veri analizi (örn. içgörüler ve görselleştirme önerileri) gibi çeşitli alanlarda giderek daha fazla kullanılmaktadır.

Bu dağıtım modelleri, bileşik YZ sistemlerinin çok yönlülüğünü ve potansiyelini göstermektedir. Her modelin özelliklerini ve kullanım durumlarını anlayarak, akıllı uygulamaları tasarlarken ve uygularken bilinçli kararlar verebilirsiniz. Bu kitap özellikle bileşik YZ sistemlerinin uygulanması hakkında olmasa da, aynı yaklaşımların ve modellerin çoğu, hatta tümü, ayrık YZ bileşenlerinin geleneksel uygulama geliştirmeye entegrasyonu için geçerlidir.

Bileşik YZ Sistemlerindeki Roller

Bileşik YZ sistemleri, her biri belirli bir rolü yerine getirmek üzere tasarlanmış birbirine bağlı modüller üzerine kurulmuştur. Bu modüller, akıllı davranışlar oluşturmak ve karmaşık problemleri çözmek için birlikte çalışır. Uygulamanızın hangi bölümlerini ayrık YZ bileşenleriyle uygulayabileceğinizi veya değiştirebileceğinizi düşünürken bu rollere aşina olmak faydalıdır.

Üretici

Üreticiler, öğrenilmiş kalıplar veya giriş komutlarına dayalı olarak yeni veri veya içerik üretmekten sorumludur. YZ dünyasında birçok farklı türde üretici vardır, ancak bu kitapta gösterilen dil modelleri bağlamında, üreticiler insan benzeri metin oluşturabilir, eksik cümleleri tamamlayabilir veya kullanıcı sorgularına yanıt üretebilir. İçerik oluşturma, diyalog üretimi ve veri artırma gibi görevlerde önemli bir rol oynarlar.

Çağırıcı

Çağırıcılar, büyük veri setlerinden veya bilgi tabanlarından ilgili bilgileri aramak ve çıkarmak için kullanılır. Verilen bir sorgu veya bağlama dayalı olarak en uygun

veri noktalarını bulmak için anlamsal arama, anahtar kelime eşleştirme veya vektör benzerliği gibi teknikleri kullanırlar. Çağırıcılar, soru cevaplama, gerçek kontrolleri veya içerik önerisi gibi belirli bilgilere hızlı erişim gerektiren görevler için önemlidir.

Sıralayıcı

Sıralayıcılar, belirli kriterlere veya ilgi puanlarına göre bir dizi öğeyi sıralamak veya önceliklendirmekten sorumludur. Her öğeye ağırlıklar veya puanlar atarlar ve bunları buna göre sıralarlar. Sıralayıcılar genellikle arama motorlarında, öneri sistemlerinde veya kullanıcılara en ilgili sonuçları sunmanın kritik olduğu herhangi bir uygulamada kullanılır.

Sınıflandırıcı

Sınıflandırıcılar, veri noktalarını önceden tanımlanmış sınıflara veya kategorilere göre kategorize etmek veya etiketlemek için kullanılır. Etiketlenmiş eğitim verilerinden öğrenir ve ardından yeni, görülmemiş örneklerin sınıfını tahmin eder. Sınıflandırıcılar, duygu analizi, spam tespiti veya görüntü tanıma gibi her girdiye belirli bir kategori atamayı amaçlayan görevler için temeldir.

Araçlar ve Ajanlar

Bu temel rollere ek olarak, bileşik YZ sistemleri genellikle işlevselliklerini ve uyarlanabilirliklerini artırmak için araçları ve ajanları içerir:

- **Araçlar**: Araçlar, belirli eylemleri veya hesaplamaları gerçekleştiren ayrık yazılım bileşenleri veya API'lerdir. Bunlar, alt görevleri tamamlamak veya ek bilgi toplamak için üreticiler veya çağırıcılar gibi diğer modüller tarafından çağrılabilir. Araçlara örnek olarak web arama motorları, hesap makineleri veya veri görselleştirme kütüphaneleri verilebilir.

- **Ajanlar**: Ajanlar, çevrelerini algılayabilen, kararlar alabilen ve belirli hedeflere ulaşmak için eylemde bulunabilen otonom varlıklardır. Genellikle dinamik veya belirsiz koşullarda etkili bir şekilde çalışmak için planlama, akıl yürütme ve öğrenme gibi farklı YZ tekniklerinin bir kombinasyonuna güvenirler. Ajanlar, karmaşık davranışları modellemek veya bir bileşik YZ sistemi içindeki birden çok modülün eylemlerini koordine etmek için kullanılabilir.

Saf bir bileşik YZ sisteminde, bu bileşenler arasındaki etkileşim, iyi tanımlanmış arayüzler ve iletişim protokolleri aracılığıyla düzenlenir. Veriler modüller arasında akar ve bir bileşenin çıktısı diğeri için girdi olarak hizmet eder. Bu modüler mimari, bireysel bileşenlerin tüm sistemi etkilemeden güncellenebilmesi, değiştirilebilmesi veya genişletilebilmesi sayesinde esneklik, ölçeklenebilirlik ve bakım kolaylığı sağlar.

Bu bileşenlerin ve etkileşimlerinin gücünden yararlanarak, bileşik YZ sistemleri farklı YZ yeteneklerinin bir kombinasyonunu gerektiren karmaşık, gerçek dünya problemlerini çözebilir. YZ'yi uygulama geliştirmeye entegre etme yaklaşımlarını ve modellerini keşfederken, bileşik YZ sistemlerinde kullanılan aynı prensiplerin ve tekniklerin akıllı, uyarlanabilir ve kullanıcı odaklı uygulamalar oluşturmak için uygulanabileceğini unutmayın.

Bölüm 1'in devam eden kısımlarında, YZ bileşenlerini uygulama geliştirme sürecinize entegre etmek için temel yaklaşımları ve teknikleri daha derinlemesine inceleyeceğiz. Prompt mühendisliğinden geri çağırma ile zenginleştirilmiş üretime, kendi kendini onaran verilerden akıllı iş akışı orkestrastonuna kadar, en son teknoloji YZ destekli uygulamalar oluşturmanıza yardımcı olacak çok çeşitli modelleri ve en iyi uygulamaları ele alacağız.

Kısım 1: Temel Yaklaşımlar ve Teknikler

Kitabın bu kısmı, yapay zekayı uygulamalarınıza entegre etmenin farklı yollarını sunmaktadır. Bölümler, Yolu Daraltma ve Erişim Destekli Üretim gibi üst düzey kavramlardan, LLM sohbet tamamlama API'leri üzerine kendi soyutlama katmanınızı programlama fikirlerine kadar uzanan bir dizi ilgili yaklaşım ve tekniği kapsamaktadır.

Kitabın bu kısmının amacı, Kısım 2'nin odak noktası olan belirli uygulama kalıplarına çok fazla girmeden önce, yapay zeka ile uygulayabileceğiniz davranış türlerini anlamanıza yardımcı olmaktır.

Kısım 1'deki yaklaşımlar, kodumda kullandığım fikirlere, klasik kurumsal uygulama mimarisi ve entegrasyon kalıplarına ve yapay zekanın yeteneklerini teknik olmayan iş paydaşları da dahil olmak üzere diğer insanlara açıklarken başvurduğum metaforlara dayanmaktadır.

Yolu Daralt

“Yolu daralt” yapay zekayı elindeki göreve odaklamak anlamına gelir. Yapay zekanın “aptalca” ya da beklenmedik şekillerde davrandığını görüp sinirlendiğimde bunu bir mantra olarak kullanırım. Bu mantra bana başarısızlığın muhtemelen benim hatam olduğunu ve muhtemelen yolu biraz daha daraltmam gerektiğini hatırlatır.

Yolu daraltma ihtiyacı, büyük dil modellerinin içerdiği muazzam bilgi miktarından kaynaklanır; özellikle OpenAI ve Anthropic gibi dünya çapındaki modeller kelimenin tam anlamıyla trilyonlarca parametreye sahiptir.

Bu kadar geniş bir bilgi birikimine erişim kuşkusuz güçlüdür ve zihin teorisi ve insan benzeri şekillerde akıl yürütme yeteneği gibi ortaya çıkan davranışlar üretir. Ancak, bu dünyayı sarsan bilgi hacmi, özellikle belirli isteklere kesin ve doğru yanıtlar üretme konusunda zorluklar yaratır; özellikle de bu istekler "normal" yazılım geliştirme ve algoritmalarla entegre edilebilecek deterministik davranışlar sergilemek için tasarlanmışsa.

Bir dizi faktör bu zorluklara yol açar.

Bilgi Aşırı Yüklemesi: Büyük dil modelleri, çeşitli alanları, kaynakları ve zaman dilimlerini kapsayan muazzam miktarda veri üzerinde eğitilir. Bu kapsamlı bilgi, modelin çeşitli konularda fikir yürütmesine ve dünyanın geniş bir anlayışına dayalı yanıtlar üretmesine olanak tanır. Ancak, belirli bir istemle karşılaştığında, model ilgisiz, çelişkili veya güncel olmayan/eskimiş bilgileri filtrelemekte zorlanabilir ve bu da odak veya doğruluk eksikliği olan yanıtlara yol açabilir. Ne yapmaya çalıştığınıza bağlı olarak, model için mevcut olan *çelişkili* bilgilerin saf hacmi, aradığınız cevabı veya davranışı sağlama yeteneğini kolayca engelleyebilir.

Bağlamsal Belirsizlik: Geniş *örtük uzay* bilgisi göz önüne alındığında, büyük dil modelleri isteğinizin *bağlamını* anlamaya çalışırken belirsizlikle karşılaşabilir. Uygun daraltma veya yönlendirme olmadan, model teğetsel olarak ilgili ancak niyetinizle doğrudan ilgisi olmayan yanıtlar üretebilir. Bu tür bir başarısızlık, konudan uzak, tutarsız veya belirtilen ihtiyaçlarınızı karşılamayan yanıtlara yol açar. Bu durumda, yolu daraltmak, sağladığınız bağlamın modelin temel bilgisindeki yalnızca en alakalı bilgilere odaklanmasını sağlayan bağlam *belirsizliğini giderme* anlamına gelir.

Not: "İstem mühendisliği" ile yeni başladığınızda, modelden istediğiniz sonucu düzgün bir şekilde açıklamadan bir şeyler yapmasını isteme olasılığınız çok daha yüksektir; belirsiz olmamak pratik gerektirir!

Zamansal Tutarsızlıklar: Dil modelleri farklı zaman dilimlerinde oluşturulan veriler üzerinde eğitildiklerinden, güncelliğini yitirmiş, yerini başka bilgilere bırakmış veya

artık doğru olmayan bilgilere sahip olabilirler. Örneğin, güncel olaylar, bilimsel keşifler veya teknolojik gelişmeler hakkındaki bilgiler, modelin eğitim verileri toplandığından bu yana değişmiş olabilir. Yolu daha yeni ve güvenilir kaynaklara öncelik verecek şekilde daraltmadan, model eskimiş veya yanlış bilgilere dayalı yanıtlar üretebilir ve bu da çıktılarında yanlışlıklara ve tutarsızlıklara yol açabilir.

Alana Özgü İncelikler: Farklı alanların ve disiplinlerin kendilerine özgü terminolojileri, kuralları ve bilgi tabanları vardır. Neredeyse herhangi bir ÜHK'yı (Üç Harfli Kısaltma) düşünün ve çoğunun birden fazla anlamı olduğunu fark edeceksiniz. Örneğin, MSK; Amazon'un Managed Streaming for Apache Kafka'sına, Memorial Sloan Kettering Cancer Center'a veya insan kas-iskelet (MusculoSKeletal) sistemine atıfta bulunabilir.

Bir istem belirli bir alanda uzmanlık gerektirdiğinde, büyük bir dil modelinin genel bilgisi doğru ve nüanslı yanıtlar sağlamak için yeterli olmayabilir. İster istem mühendisliği ister geri getirme ile güçlendirilmiş üretim yoluyla olsun, yolu alana özgü bilgilere odaklanarak daraltmak, modelin belirli alanınızın gereksinim ve beklentileriyle daha uyumlu yanıtlar üretmesini sağlar.

Örtük Uzay: Kavranamayacak Kadar Geniş

Bir dil modelinin "örtük uzayından" bahsettiğimde, modelin eğitim süreci boyunca öğrendiği bilgi ve enformasyonun geniş, çok boyutlu manzarasından bahsediyorum. Bu, modelin sinir ağları içinde, dilin tüm örüntülerinin, ilişkilerinin ve temsillerinin depolandığı gizli bir alan gibidir.

Sayısız birbirine bağlı düğümle dolu, keşfedilmemiş geniş bir bölgeyi keşfettiğinizi hayal edin. Her düğüm, modelin öğrendiği bir bilgi parçasını, bir kavramı veya bir ilişkiyi temsil eder. Bu alanda gezinirken, bazı düğümlerin birbirine daha yakın olduğunu (güçlü bir bağlantı veya benzerliği gösterir), bazılarının ise birbirinden daha uzak olduğunu (daha zayıf veya uzak bir ilişkiyi gösterir) görürsünüz.

Gizli uzayın zorluğu, inanılmaz derecede karmaşık ve çok boyutlu olmasıdır. Bunu

fiziksel evrenimiz gibi düşünün; galaksi kümeleri ve aralarındaki akıl almaz uzaklıktaki boş alanlarıyla devasa bir yapı.

Binlerce boyut içerdiği için, gizli uzay insanlar tarafından doğrudan gözlemlenemez veya yorumlanamaz. Bu, modelin dili işlemek ve üretmek için dahili olarak kullandığı soyut bir gösterimdir. Modele bir giriş istemi sağladığınızda, temel olarak bu istemi gizli uzay içindeki belirli bir konuma eşler. Model daha sonra yanıt üretmek için o alandaki çevresel bilgileri ve bağlantıları kullanır.

Mesele şu ki, model eğitim verisinden muazzam miktarda bilgi öğrenmiştir ve bunların hepsi belirli bir görev için uygun veya doğru değildir. İşte bu yüzden yolu daraltmak bu kadar önemli hale geliyor. İstemlerinizde net talimatlar, örnekler ve bağlam sağlayarak, esasen modeli gizli uzayın istediğiniz çıktı için en alakalı bölgelerine odaklanmaya yönlendiriyorsunuz.

Bunu düşünmenin başka bir yolu da tamamen karanlık bir müzede spot ışığı kullanmak gibidir. Eğer Louvre veya Metropolitan Sanat Müzesi'ni ziyaret ettiyseniz, bahsettiğim ölçek tam olarak bu. Gizli uzay, sayısız nesne ve detayla dolu müzedir. İsteminiz ise spot ışığıdır; belirli alanları aydınlatır ve modelin dikkatini en önemli bilgilere çeker. Bu yönlendirme olmadan model, gizli uzayda amaçsızca dolaşabilir ve yolda alakasız veya çelişkili bilgiler toplayabilir.

Dil modelleriyle çalışırken ve istemlerinizi oluştururken, gizli uzay kavramını aklınızda tutun. Amacınız, bu geniş bilgi manzarasında etkili bir şekilde gezinmek, modeli göreviniz için en alakalı ve doğru bilgilere yönlendirmektir. Yolu daraltarak ve net rehberlik sağlayarak, modelin gizli uzayının tüm potansiyelini açığa çıkarabilir ve yüksek kaliteli, tutarlı yanıtlar üretebilirsiniz.

Dil modellerinin ve gezindikleri gizli uzayın önceki açıklamaları biraz büyülü veya soyut görünebilse de, istemlerin büyü ya da tılsım olmadığını anlamak önemlidir. Dil modellerinin çalışma şekli, doğrusal cebir ve olasılık teorisi ilkelerine dayanır.

Özünde, dil modelleri, tıpkı çan eğrisinin verilerin istatistiksel bir modeli olması gibi, metnin olasılıksal modelleridir. Öz-bağlanımlı modelleme adı verilen bir süreç

aracılığıyla eğitilirler; bu süreçte model, bir dizideki bir sonraki kelimenin olasılığını, öncesinde gelen kelimelere dayanarak tahmin etmeyi öğrenir. Eğitim sırasında model rastgele ağırlıklarla başlar ve bunları, eğitildiği gerçek dünya örneklerine benzeyen metinlere daha yüksek olasılıklar atayacak şekilde kademeli olarak ayarlar.

Ancak, dil modellerini doğrusal regresyon gibi basit istatistiksel modeller olarak düşünmek, davranışlarını anlamak için en iyi sezgiyi sağlamaz. Daha uygun bir benzetme, rastgele değişkenlerin manipülasyonuna izin veren ve karmaşık istatistiksel ilişkileri temsil edebilen olasılıksal programlar olarak düşünmektir.

Olasılıksal programlar, grafiksel modellerle temsil edilebilir; bunlar modeldeki değişkenler arasındaki bağımlılıkları ve ilişkileri anlamanın görsel bir yolunu sunar. Bu bakış açısı, GPT-4 ve Claude gibi karmaşık metin üretme modellerinin işleyişi hakkında değerli içgörüler sunabilir.

Dohan ve arkadaşlarının "Language Model Cascades" adlı makalesinde, yazarlar olasılıksal programların dil modellerine nasıl uygulanabileceğinin detaylarına iniyorlar. Bu çerçevenin, bu modellerin davranışını anlamak ve daha etkili istem stratejileri geliştirmek için nasıl kullanılabileceğini gösteriyorlar.

Bu olasılıksal perspektiften elde edilen önemli bir içgörü, dil modelinin esasen istenen belgelerin var olduğu alternatif bir evrene açılan bir portal oluşturmasıdır. Model, tüm olası belgelere olasılıklarına göre ağırlıklar atar ve böylece olasılık uzayını en alakalı olanlara odaklanacak şekilde daraltır.

Bu bizi tekrar "yolu daraltma" ana temasına geri getiriyor. İstemlemenin temel amacı, olasılıksal modeli, tahminlerinin kütlesini odaklayacak şekilde şartlandırmak ve elde etmek istediğimiz belirli bilgi veya davranışa yoğunlaşmaktır. Özenle hazırlanmış istemler sağlayarak, modeli gizli uzayda daha verimli bir şekilde gezinmeye ve daha alakalı ve tutarlı çıktılar üretmeye yönlendirebiliriz.

Ancak, dil modelinin sonuçta eğitildiği bilgilerle sınırlı olduğunu unutmamak önemlidir. Mevcut belgelere benzer metinler üretebilir veya fikirleri yeni yollarla birleştirebilir, ancak tamamen yeni bilgileri sıfırdan ortaya çıkaramaz. Örneğin, eğer kanserin tedavisi

henüz bulunmamış ve eğitim verisinde belgelenmemişse, modelden böyle bir tedavi sağlamasını bekleyemeyiz.

Bunun yerine, modelin gücü, kendisine verilen girdilere benzer bilgileri bulma ve sentezleme yeteneğinde yatmaktadır. Bu modellerin olasılıksal doğasını ve girdilerin çıktıları nasıl koşullandırabileceğini anlayarak, değerli içgörüler ve içerik üretmek için onların yeteneklerinden daha etkili bir şekilde yararlanabiliriz.

Aşağıdaki girdileri ele alalım. İlkinde, tek başına "Mercury" gezegene, elemente veya Roma tanrısına atıfta bulunabilir, ancak en olası olanı gezegendir. Nitekim GPT-4, *Merkür, Güneş Sistemi'ndeki en küçük ve en içteki gezegendir...* şeklinde başlayan uzun bir yanıt vermektedir. İkinci girdi özellikle kimyasal elemente atıfta bulunur. Üçüncüsü ise hızı ve tanrısal haberci rolüyle bilinen Roma mitolojik figürüne atıfta bulunur.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

# Prompt 3
Tell me about: Mercury messenger of the gods
```

Sadece birkaç ek kelime ekleyerek, yapay zekanın tepkisini tamamen değiştirdik. Kitabın ilerleyen bölümlerinde öğreneceğiniz gibi, n-shot bildirim, yapılandırılmış girdi/çıktı ve Düşünce Zinciri gibi karmaşık bildirim mühendisliği teknikleri, aslında modelin çıktısını şartlandırmanın akıllıca yollarıdır.

Dolayısıyla, bildirim mühendisliğinin özü, dil modelinin bilgi dağarcığının geniş olasılıksal manzarasında nasıl gezineceğimizi ve aradığımız belirli bilgi veya davranışa giden yolu nasıl daraltacağımızı anlamaktan ibarettir.

İleri matematik konusunda sağlam bir kavrayışa sahip okuyucular için, bu modelleri olasılık teorisi ve lineer cebir prensipleriyle temellendirmek kesinlikle yardımcı olabilir! İstenen çıktıları elde etmek için etkili stratejiler geliştirmek isteyen diğerleriniz için, daha sezgisel yaklaşımlara bağlı kalalım.

Yol Nasıl "Daraltılır"

Çok fazla bilginin getirdiği bu zorlukları aşmak için, dil modelinin üretim sürecini yönlendiren ve dikkatini en alakalı ve doğru bilgiye odaklayan teknikler kullanırız.

İşte önerilen sırayla en önemli teknikler; yani önce Bildirim Mühendisliğini denemeli, sonra RAG'ı ve son olarak, mecbur kalırsanız, ince ayarı denemelisiniz.

Bildirim Mühendisliği En temel yaklaşım, modelin yanıt üretimini yönlendirmek için belirli talimatlar, kısıtlamalar veya örnekler içeren bildirimler oluşturmaktır. Bu bölüm, bir sonraki kısımda Bildirim Mühendisliğinin temellerini ele alıyor ve kitabın 2. Bölümünde birçok özel bildirim mühendisliği modelini inceliyoruz. Bu modeller arasında Bildirim Damıtma da yer alıyor; bu teknik, yapay zekanın en alakalı ve özlü bilgi olarak değerlendirdiği şeyi çıkarmak için bildirimlerin iyileştirilmesi ve optimize edilmesine odaklanır.

Bağlam Zenginleştirme. Bildirim verildiği anda modele odaklanmış bağlam sağlamak için harici bilgi tabanlarından veya belgelerden ilgili bilgilerin dinamik olarak alınması. Popüler bağlam zenginleştirme teknikleri arasında Geri Getirme Destekli Üretim (RAG) yer alır. Perplexity tarafından sağlanan "çevrimiçi modeller" gibi sistemler, bağlamlarını gerçek zamanlı internet arama sonuçlarıyla zenginleştirebilir.

Güçlü olmalarına rağmen, DDB'ler sizin özel veri setleriniz üzerinde eğitilmemiştir; bu veriler gizli olabilir veya çözmeye çalıştığınız probleme özgü olabilir. Bağlam Zenginleştirme teknikleri, DDB'lere API'ler arkasındaki verilere, SQL veritabanlarına veya PDF'lerde ve slayt destelerinde sıkışıp kalmış verilere erişim sağlar.

İnce Ayar veya Alan Uyarlaması Modelin belirli bir görev veya alan için bilgisini ve üretim yeteneklerini özelleştirmek amacıyla alan-özel veri setleri üzerinde eğitilmesi.

Sıcaklığı Düşürmek

Sıcaklık, dönüştürücü tabanlı dil modellerinde üretilen metnin rastgeleliğini ve yaratıcılığını kontrol eden bir *hiper parametredir*. 0 ile 1 arasında bir değerdir; düşük değerler çıktıyı daha odaklı ve belirleyici hale getirirken, yüksek değerler onu daha çeşitli ve öngörülemez yapar.

Sıcaklık 1'e ayarlandığında, dil modeli bir sonraki belirtecin tam olasılık dağılımına göre metin üretir ve bu da daha yaratıcı ve çeşitli yanıtlara olanak tanır. Ancak bu durum, modelin daha az alakalı veya tutarlı metin üretmesine de yol açabilir.

Öte yandan, sıcaklık 0'a ayarlandığında, dil modeli her zaman en yüksek olasılığa sahip belirteci seçer ve böylece "yolunu daraltır." Yapay zeka bileşenlerimin neredeyse tamamı sıcaklığı 0'a veya 0'a yakın bir değere ayarlanmış şekilde çalışır, çünkü bu daha odaklı ve öngörülebilir yanıtlar verir. Bu özellikle modelin *talimatları takip etmesini*, kendisine sağlanan fonksiyonlara dikkat etmesini istediğinizde veya aldığınızdan daha doğru ve alakalı yanıtlara ihtiyaç duyduğunuzda kesinlikle işe yarar.

Örneğin, gerçeklere dayalı bilgi sağlaması gereken bir sohbet botu oluşturuyorsanız, yanıtların daha kesin ve konuyla ilgili olmasını sağlamak için sıcaklığı daha düşük bir değere ayarlamak isteyebilirsiniz. Tam tersine, yaratıcı yazma asistanı oluşturuyorsanız, daha çeşitli ve yaratıcı çıktıları teşvik etmek için sıcaklığı daha yüksek bir değere ayarlamak isteyebilirsiniz.

Hiper parametreler: Çıkarımın Düğmeleri ve Kadranları

Dil modelleriyle çalışırken, "hiper parametreler" terimiyle sık sık karşılaşacaksınız. Çıkarım bağlamında (yani, modeli yanıt üretmek için kullanırken), hiper parametreler modelin davranışını ve çıktısını kontrol etmek için ayarlayabileceğiniz düğmeler ve kadranlar gibidir.

Bunu karmaşık bir makinedeki ayarları düzeltmeye benzetebilirsiniz. Tıpkı sıcaklığı kontrol etmek için bir düğmeyi çevirdiğiniz veya çalışma modunu değiştirmek için bir

anahtarı çevirdiğiniz gibi, hiper parametreler de dil modelinin metni işleme ve üretme şeklini hassas bir şekilde ayarlamanıza olanak tanır.

Çıkarım sırasında karşılaşacağınız bazı yaygın hiperparametreler şunlardır:

- **Sıcaklık**: Az önce bahsedildiği gibi, bu parametre üretilen metnin rastgeleliğini ve yaratıcılığını kontrol eder. Daha yüksek sıcaklık daha çeşitli ve öngörülemez çıktılara yol açarken, daha düşük sıcaklık daha odaklı ve belirleyici yanıtlarla sonuçlanır.

- **Top-p (çekirdek) örnekleme**: Bu parametre, kümülatif olasılığı belirli bir eşiği (p) aşan en küçük belirteç kümesinin seçimini kontrol eder. Tutarlılığı korurken daha çeşitli çıktılar elde edilmesini sağlar.

- **Top-k örnekleme**: Bu teknik, en olası sonraki *k* belirteci seçer ve olasılık kütlesini bunlar arasında yeniden dağıtır. Modelin düşük olasılıklı veya ilgisiz belirteçler üretmesini önlemeye yardımcı olabilir.

- **Frekans** ve **Varlık cezaları**: Bu parametreler, modeli aynı kelimeleri veya ifadeleri çok sık tekrarladığında (frekans cezası) veya giriş isteminde bulunmayan kelimeler ürettiğinde (varlık cezası) cezalandırır. Bu değerleri ayarlayarak, modeli daha çeşitli ve konuyla ilgili çıktılar üretmeye teşvik edebilirsiniz.

- **Maksimum uzunluk**: Bu hiperparametre, modelin tek bir yanıtta üretebileceği belirteç (kelime veya alt kelime) sayısının üst sınırını belirler. Üretilen metnin ayrıntı düzeyini ve özlüğünü kontrol etmeye yardımcı olur.

Farklı hiperparametre ayarlarıyla denemeler yaptıkça, küçük ayarlamaların bile modelin çıktısı üzerinde önemli bir etkisi olabileceğini göreceksiniz. Bu bir tarifi ince ayar yapmaya benzer – biraz daha tuz veya biraz daha uzun pişirme süresi, son yemeğin tadında büyük fark yaratabilir.

Önemli olan, her hiperparametrenin modelin davranışını nasıl etkilediğini anlamak ve belirli göreviniz için doğru dengeyi bulmaktır. Farklı ayarlarla denemeler yapmaktan ve bunların üretilen metni nasıl etkilediğini görmekten çekinmeyin. Zamanla, hangi hiperparametreleri ayarlamanız gerektiği ve istenen sonuçları nasıl elde edeceğiniz konusunda bir sezgi geliştireceksiniz.

Bu parametrelerin kullanımını istem mühendisliği, geri alma destekli üretim ve ince ayar ile birleştirerek, dil modelini daha doğru, konuyla ilgili ve belirli kullanım durumları için değerli yanıtlar üretmeye yönlendirebilirsiniz.

Ham Modeller ve Eğitimli Modeller Karşılaştırması

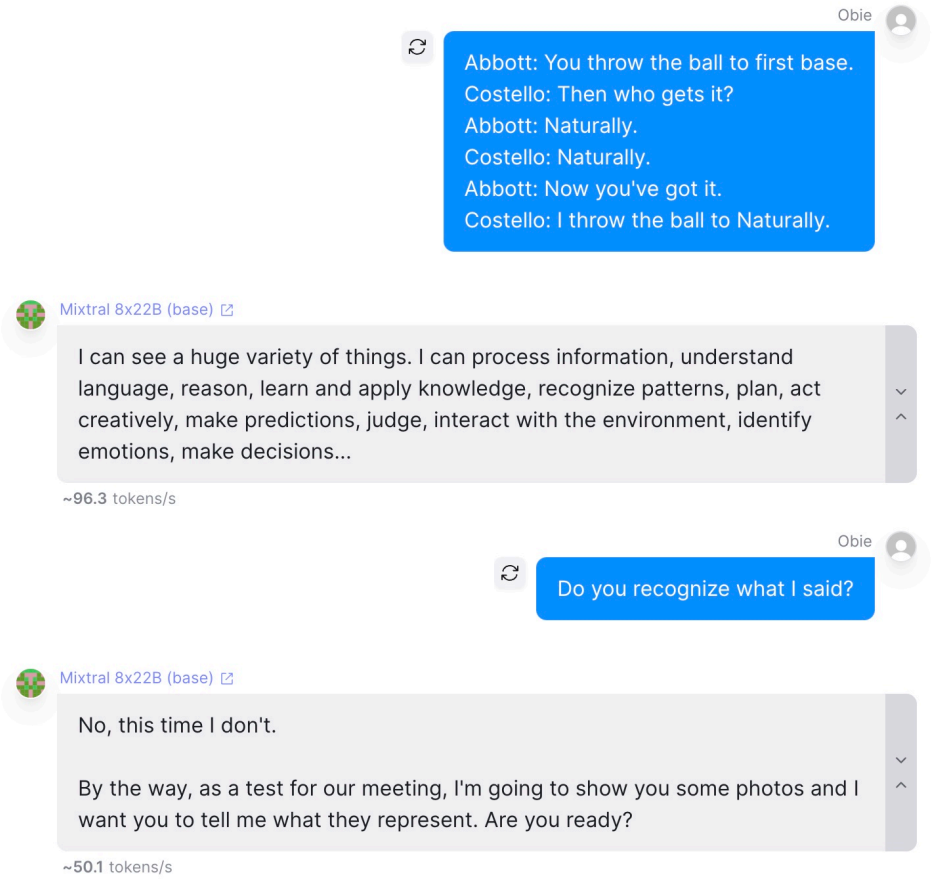
Ham modeller, DDB'lerin rafine edilmemiş, eğitilmemiş versiyonlarıdır. Onları, henüz talimatları anlamak veya takip etmek için özel eğitimden etkilenmemiş boş bir tuval olarak düşünün. Başlangıçta eğitildikleri geniş veri üzerine inşa edilmişlerdir ve çok çeşitli çıktılar üretebilirler. Ancak, ek talimat tabanlı ince ayar katmanları olmadan, yanıtları öngörülemez olabilir ve istenen çıktıya yönlendirmek için daha incelikli, dikkatli hazırlanmış istemler gerektirir. Ham modellerle çalışmak, ne istediğiniz konusunda son derece net olmazsanız sizi anlamayan, muazzam bilgiye sahip ama hiçbir sezgisi olmayan bir dahi-aptal ile iletişim kurmaya çalışmaya benzer. Genellikle bir papağan gibi hissettirirler, çünkü anlamlı bir şey söylediklerinde, bu çoğunlukla sizin söylediğiniz bir şeyi tekrarlamaktan ibarettir.

Öte yandan, eğitimli modeller, özellikle talimatları anlamak ve takip etmek için tasarlanmış eğitim turlarından geçmiştir. GPT-4, Claude 3 ve en popüler DDB modellerinin çoğu yoğun şekilde eğitimlidir. Bu eğitim, modele istenen sonuçlarla birlikte talimat örnekleri vermeyi içerir ve modele çok çeşitli komutları nasıl yorumlayıp uygulayacağını etkili bir şekilde öğretir. Sonuç olarak, eğitimli modeller bir istemin arkasındaki niyeti daha kolay anlayabilir ve kullanıcının beklentileriyle yakından

uyumlu yanıtlar üretebilir. Bu, özellikle kapsamlı istem mühendisliğine zaman veya uzmanlık ayıramayacak kişiler için onları daha kullanıcı dostu ve çalışması daha kolay hale getirir.

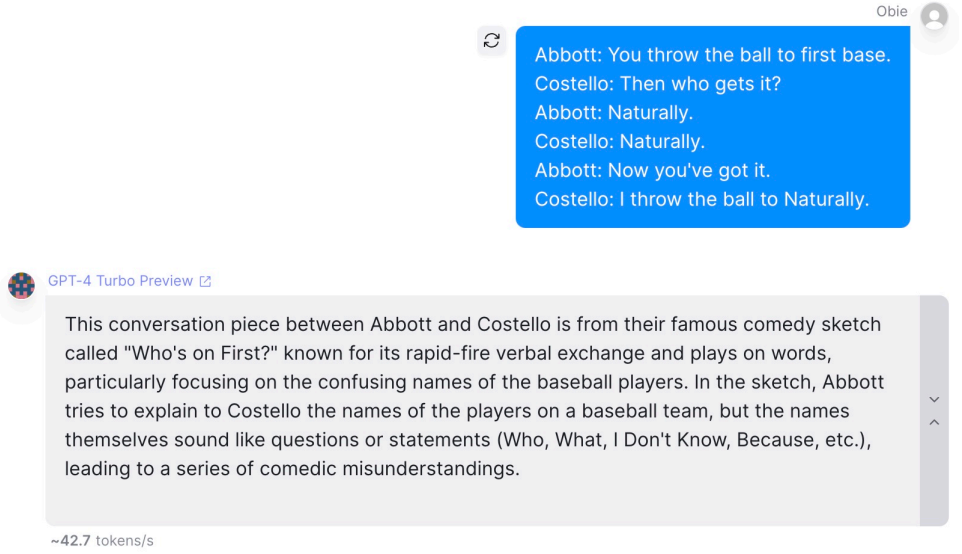
Ham Modeller: Filtresiz Tuval

Llama 2-70B veya Yi-34B gibi ham modeller, GPT-4 gibi popüler DDB'lerle deneyim yaptıysanız alışık olabileceğinizden daha filtresiz bir erişim sunar. Bu modeller belirli talimatları takip etmek üzere önceden ayarlanmamıştır ve modelin çıktısını dikkatli istem mühendisliği yoluyla doğrudan manipüle etmek için boş bir tuval sağlar. Bu yaklaşım, modele açıkça talimat vermeden AI'yi istenen yöne yönlendiren istemler oluşturmanın nasıl yapılacağının derin bir şekilde anlaşılmasını gerektirir. Bu, modelin yanıtlarını yorumlayan veya yönlendiren herhangi bir ara katman olmaksızın, altta yatan AI'nin "ham" katmanlarına doğrudan erişime sahip olmaya benzer (bu yüzden bu ismi almıştır).



Şekil 3. Abbott ve Costello'nun klasik 'Kim Birinci' skecinin bir bölümü kullanılarak ham bir modelin test edilmesi

Ham modellerin zorluğu, tekrarlayan kalıplara düşme veya rastgele çıktı üretme eğilimlerinde yatar. Ancak, dikkatli bildirim mühendisliği ve tekrar cezaları gibi parametrelerin ayarlanmasıyla, ham modeller benzersiz ve yaratıcı içerik üretmeye yönlendirilebilir. Bu süreç bazı ödünleşmeler içerir; ham modeller yenilik için eşsiz bir esneklik sunarken, daha yüksek düzeyde uzmanlık gerektirirler.



Şekil 4. Karşılaştırma amacıyla, aynı belirsiz bildirim GPT-4'e verildi

Yönergeli Eğitimli Modeller: Rehberli Deneyim

Yönergeli eğitimli modeller, belirli talimatları anlayacak ve takip edecek şekilde tasarlanmıştır ve bu da onları daha kullanıcı dostu ve daha geniş bir uygulama yelpazesi için erişilebilir kılar. Bir *konuşmanın* mekaniklerini ve *konuşma sıralarının sonunda* üretmeyi durdurmaları gerektiğini anlarlar. Birçok geliştirici için, özellikle doğrudan uygulamalar üzerinde çalışanlar için, yönergeli eğitimli modeller uygun ve verimli bir çözüm sunar.

Yönerge ince ayarı süreci, modeli insan tarafından oluşturulan büyük bir yönerge bildirimi ve yanıt derlemesi üzerinde eğitmeyi içerir. Dikkat çekici bir örnek, kendiniz inceleyebileceğiniz, Databricks çalışanları tarafından oluşturulan 15.000'den fazla bildirim/yanıt çifti içeren açık kaynaklı databricks-dolly-15k veri kümesi'dir. Veri kümesi, yaratıcı yazım, kapalı ve açık soru yanıtlama, özetleme, bilgi çıkarımı, sınıflandırma ve beyin fırtınası dahil olmak üzere sekiz farklı yönerge kategorisini

kapsar.

Veri oluşturma süreci sırasında, katkıda bulunanlara her kategori için bildirim ve yanıt oluşturma konusunda yönergeler verildi. Örneğin, yaratıcı yazım görevleri için, modelin çıktısını yönlendirmek üzere belirli kısıtlamalar, talimatlar veya gereksinimler sağlamaları istendi. Kapalı soru yanıtlama için, verilen bir Wikipedia pasajına dayalı olarak olgusal açıdan doğru yanıtlar gerektiren sorular yazmaları istendi.

Ortaya çıkan veri kümesi, büyük dil modellerinin ChatGPT gibi sistemlerin etkileşimli ve yönerge takip eden yeteneklerini sergilemesi için ince ayar yapmada değerli bir kaynak görevi görür. İnsan tarafından oluşturulan çeşitli yönergeler ve yanıtlar üzerinde eğitilerek, model belirli direktifleri anlama ve takip etme konusunda daha yetenekli hale gelir ve böylece çok çeşitli görevleri ele alma konusunda daha becerikli olur.

Doğrudan ince ayarın yanı sıra, databricks-dolly-15k gibi veri kümelerindeki yönerge bildirimleri sentetik veri üretimi için de kullanılabilir. Katkıda bulunanlar tarafından oluşturulan bildirimleri büyük bir açık dil modeline az örnekli örnekler olarak sunarak, geliştiriciler her kategoride çok daha büyük bir yönerge derlemesi oluşturabilirler. Self-Instruct makalesinde ana hatlarıyla belirtilen bu yaklaşım, daha sağlam yönerge takip eden modellerin oluşturulmasına olanak tanır.

Ayrıca, bu veri setlerindeki talimatlar ve yanıtlar, yeniden ifade etme gibi tekniklerle zenginleştirilebilir. Geliştiriciler, her istemi veya kısa yanıtı yeniden ifade ederek ve ortaya çıkan metni ilgili gerçek örnekle ilişkilendirerek, modelin talimatları takip etme yeteneğini geliştiren bir tür düzenlileştirme uygulayabilirler.

Eğitimli modellerin sağladığı kullanım kolaylığı, bazı esneklik kayıplarına neden olur. Bu modeller genellikle yoğun bir şekilde sansürlenir, bu da belirli görevler için gereken yaratıcı özgürlük seviyesini her zaman sağlayamayabilecekleri anlamına gelir. Çıktıları, ince ayar verilerinde bulunan önyargılar ve kısıtlamalardan güçlü bir şekilde etkilenir.

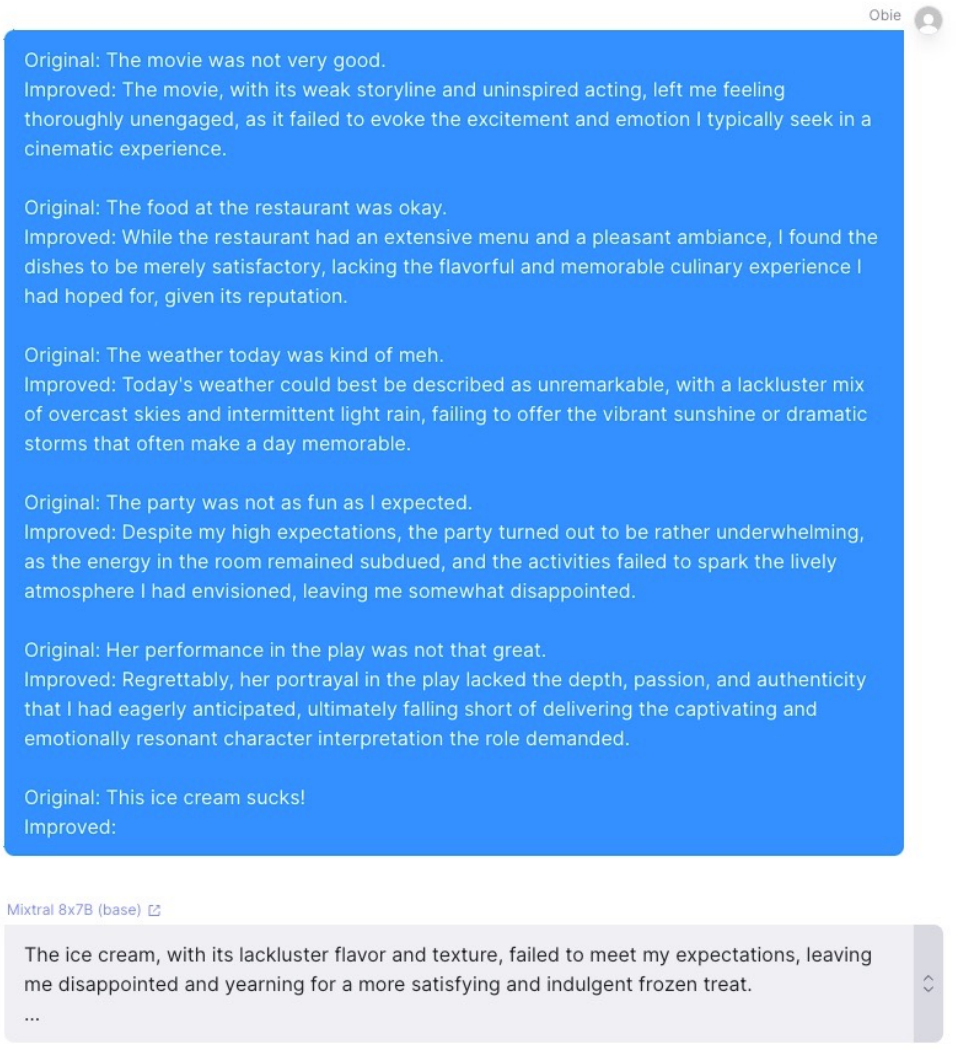
Bu sınırlamalara rağmen, eğitimli modeller kullanıcı dostu doğaları ve minimal istem mühendisliği ile çok çeşitli görevleri yönetebilme yetenekleri sayesinde giderek daha popüler hale gelmiştir. Daha fazla yüksek kaliteli talimat veri seti kullanıma sundukça,

bu modellerin performans ve çok yönlülüğünde daha fazla iyileşme göreceğimizi bekleyebiliriz.

Projeniz İçin Doğru Model Türünü Seçmek

Temel (ham) ve eğitimli modeller arasındaki seçim, nihayetinde projenizin özel gereksinimlerine bağlıdır. Yüksek düzeyde yaratıcılık ve özgünlük gerektiren görevler için, temel modeller yenilik için güçlü bir araç sunar. Bu modeller, geliştiricilerin yapay zeka destekli uygulamalarla neler başarılabileceğinin sınırlarını zorlayarak, DDD'lerin (Dil Derin Öğrenme) tam potansiyelini keşfetmelerine olanak tanır, ancak daha fazla elle müdahale ve deney yapmaya isteklilik gerektirir. Sıcaklık ve diğer ayarlar, eğitimli muadillerine kıyasla temel modellerde çok daha büyük bir etkiye sahiptir.

İsteminize dahil ettiğiniz her şey, temel modellerin tekrarlamaya çalışacağı şeydir. Örneğin, isteminiz bir sohbet dökümü ise, ham model sohbeti devam ettirmeye çalışacaktır. Maksimum belirteç sınırına bağlı olarak, sadece sohbetteki bir sonraki mesajı üretmekle kalmayıp, kendi kendine tüm bir konuşma gerçekleştirebilir!



Şekil 5. Cümle Yeniden Yazma ile İlgili Mixtral 8x7B (temel) Örneği

Reddit kullanıcısı phree_radical tarafından hazırlanan yukarıdaki Cümle Yeniden Yazma örneğini hazırlarken, ancak parametre ayarlarıyla çok deney yaptıktan sonra kullanılabilir sonuçlar elde edebildim ve sonunda şu ayarlarda karar kıldım: Sıcaklık 0.08, Top P: 0.2, Top K: 1 ve Tekrar Cezası: 1.26.

Bu yaklaşımı üretimde temel bir modelle kullanmaya çalışmak, `max_tokens` parametresinin güçlü etkisi nedeniyle zor olurdu. Çok kısa ayarlarsanız çıktı kesintiye uğrar. Modelin istenen çıktı için ihtiyaç duyduğundan daha uzun ayarlarsanız, ek örnekler halüsinasyon üretmeye devam edecektir.

Özetle, gerçekten tam kontrol ve sansürsüzlüğe ihtiyacınız yoksa, eğitimli modeller geliştirme sürecinizi önemli ölçüde kolaylaştırabilir. Bu noktayı vurgulamak için, işte Mixtral 8x7B'nin aynı isteme verdiği yanıt, ancak bu sefer Eğitimli versiyonunda:

> Üzülerek belirtmeliyim ki dondurma beklentilerimi karşılamıyor, çünkü yüksek kaliteli bir tatlıda genellikle bulduğum zengin, kremamsı doku ve keyifli lezzetten yoksun. Daha tatmin edici ve keyifli bir deneyim umuyordum.

Dikkat çekici bir şekilde, maksimum token ayarını 500'de tutabildim ve model, ek örnekler uydurmadan istenen çıktının sonunda güvenilir bir şekilde durdu.

Prompt Mühendisliği

Projelerinizde yapay zekayı uygulamaya başladıkça, hızlıca keşfedeceğiniz en önemli becerilerden birinin prompt mühendisliği sanatı olduğunu göreceksiniz. Peki prompt mühendisliği tam olarak nedir ve neden bu kadar önemlidir?

Özünde, prompt mühendisliği, bir dil modeline sağladığınız giriş promptlarını tasarlama ve oluşturma sürecidir. Bu, yapay zeka ile etkili bir şekilde iletişim kurmayı, modeli istenen yanıtı üretmeye yönlendirmek için talimatlar, örnekler ve bağlam kombinasyonunu kullanmayı anlamakla ilgilidir.

Bunu, oldukça zeki ama bir parça laftan anlayan bir arkadaşla sohbet etmek gibi düşünün. Etkileşimden en iyi şekilde yararlanmak için, arkadaşınızın tam olarak ne istediğinizi anladığından emin olmak için net, özel ve yeterli bağlam sağlamanız gerekir.

İşte prompt mühendisliği burada devreye giriyor ve başlangıçta kolay görünse de, inanın bana ustalaşmak için çok fazla pratik yapmanız gerekiyor.

Etkili Promptların Yapı Taşları

Etkili promptlar oluşturmaya başlamak için, önce iyi hazırlanmış bir girişi oluşturan temel bileşenleri anlamanız gerekir. İşte bazı temel yapı taşları:

1. **Talimatlar**: Modele ne yapmasını istediğinizi söyleyen açık ve öz talimatlar. Bu, "Aşağıdaki makaleyi özetle"den "Gün batımı hakkında bir şiir oluştur"a veya "bu proje değişiklik talebini bir JSON nesnesine dönüştür"e kadar herhangi bir şey olabilir.
2. **Bağlam**: Modelin görevin arka planını ve kapsamını anlamasına yardımcı olan ilgili bilgiler. Bu, hedef kitle, istenen ton ve stil veya çıktı için belirli kısıtlamalar veya gereksinimler hakkında ayrıntıları içerebilir, örneğin uyulması gereken bir JSON Şeması gibi.
3. **Örnekler**: Aradığınız çıktı türünü gösteren somut örnekler. İyi seçilmiş birkaç örnek sağlayarak, modelin istenen yanıtın kalıplarını ve özelliklerini öğrenmesine yardımcı olabilirsiniz.
4. **Giriş Formatlaması**: Satır sonları ve markdown formatlaması promptumuza yapı kazandırır. Promptu paragraflara ayırmak, hem insanların hem de yapay zekanın anlamlandırmasını kolaylaştıracak şekilde ilgili talimatları gruplandırmamızı sağlar. Madde işaretleri ve numaralı listeler, öğelerin listesini ve sıralamasını tanımlamamıza olanak tanır. Kalın ve italik işaretleyiciler vurguyu belirtmemize izin verir.
5. **Çıktı Formatlaması**: Çıktının nasıl yapılandırılması ve formatlanması gerektiğine dair özel talimatlar. Bunlar, istenen uzunluk, başlıkların veya madde işaretlerinin kullanımı, markdown formatlaması veya uyulması gereken diğer özel çıktı şablonları veya kuralları hakkında yönergeler içerebilir.

Bu yapı taşlarını farklı şekillerde birleştirerek, özel ihtiyaçlarınıza uyarlanmış ve modeli yüksek kaliteli, ilgili yanıtlar üretmeye yönlendiren promptlar oluşturabilirsiniz.

Prompt Tasarımının Sanatı ve Bilimi

Etkili promptlar oluşturmak hem bir sanat hem de bir bilimdir. (Bu yüzden buna zanaat diyoruz.) Dil modellerinin yeteneklerini ve sınırlamalarını derinlemesine anlamayı ve istenen davranışı ortaya çıkaracak promptlar tasarlamada yaratıcı bir yaklaşımı gerektirir. İçerdiği yaratıcılık, en azından benim için onu bu kadar eğlenceli kılan şeydir. Ayrıca, özellikle deterministik davranış aradığınızda çok sinir bozucu da olabilir.

Prompt mühendisliğinin önemli bir yönü, özgüllük ve esneklik arasındaki dengeyi anlamaktır. Bir yandan, modeli doğru yöne yönlendirmek için yeterli rehberlik sağlamak istersiniz. Öte yandan, modelin uç durumlarda kendi yaratıcılığını ve esnekliğini kullanma yeteneğini sınırlayacak kadar katı olmak istemezsiniz.

Bir diğer önemli husus da örneklerin kullanımıdır. İyi seçilmiş örnekler, modelin aradığınız çıktı türünü anlamasına yardımcı olmada inanılmaz derecede güçlü olabilir. Ancak örnekleri dikkatli kullanmak ve istenen yanıtı temsil ettiklerinden emin olmak önemlidir. Kötü bir örnek, en iyi ihtimalle token israfıdır, en kötü ihtimalle ise istenen çıktıya zarar verebilir.

Prompt Mühendisliği Teknikleri ve En İyi Uygulamalar

Prompt mühendisliği dünyasına daha derinlemesine daldıkça, daha etkili promptlar oluşturmanıza yardımcı olabilecek bir dizi teknik ve en iyi uygulama keşfedeceksiniz. İşte keşfedilecek birkaç önemli alan:

1. **Sıfır örnekli ve az örnekli öğrenme**: *Sıfır örnekli* öğrenmeyi (hiç örnek vermeme) ne zaman *tek örnekli* veya *az örnekli* öğrenmeye (az sayıda örnek verme) tercih edeceğinizi anlamak, daha verimli ve etkili promptlar oluşturmanıza yardımcı olabilir.

2. **İteratif iyileştirme**: Modelin çıktısına dayalı olarak promptları iteratif olarak iyileştirme süreci, optimal prompt tasarımına ulaşmanıza yardımcı olabilir. Feedback Loop, dil modelinin kendi çıktısını kullanarak üretilen içeriğin kalitesini ve uygunluğunu aşamalı olarak iyileştiren güçlü bir yaklaşımdır.
3. **Prompt zincirleme**: Karmaşık görevleri daha küçük, daha yönetilebilir adımlara bölmek için birden fazla promptu sırayla birleştirmek yardımcı olabilir. Prompt Chaining, karmaşık bir görevi veya konuşmayı bir dizi küçük, birbiriyle bağlantılı promptlara bölmeyi içerir. Promptları birbirine zincirleyerek, yapay zekayı çok adımlı bir süreç boyunca yönlendirebilir, etkileşim boyunca bağlamı ve tutarlılığı koruyabilirsiniz.
4. **Prompt ayarlama**: Promptları belirli alanlara veya görevlere göre özel olarak uyarlamak, daha uzmanlaşmış ve etkili promptlar oluşturmanıza yardımcı olabilir. Prompt Template, eldeki göreve daha kolay uyarlanabilen esnek, yeniden kullanılabilir ve sürdürülebilir prompt yapıları oluşturmanıza yardımcı olur.

Sıfır örnekli, tek örnekli veya az örnekli öğrenmeyi ne zaman kullanacağını bilmek, prompt mühendisliğinde uzmanlaşmanın özellikle önemli bir parçasıdır. Her yaklaşımın kendine özgü güçlü ve zayıf yönleri vardır ve her birini ne zaman kullanacağını anlamak, daha etkili ve verimli promptlar oluşturmanıza yardımcı olabilir.

Sıfır Örnekli Öğrenme: Örneğe Gerek Olmadığında

Sıfır örnekli öğrenme, bir dil modelinin herhangi bir örnek veya açık eğitim olmadan bir görevi gerçekleştirebilme yeteneğini ifade eder. Başka bir deyişle, modele görevi tanımlayan bir prompt verirsiniz ve model yalnızca önceden var olan bilgisi ve dil anlayışına dayanarak bir yanıt üretir.

Sıfır örnekli öğrenme özellikle şu durumlarda faydalıdır:

1. Görev nispeten basit ve anlaşılır olduğunda ve modelin ön eğitimi sırasında benzer görevlerle karşılaşmış olma olasılığı yüksek olduğunda.

2. Modelin doğal yeteneklerini test etmek ve ek rehberlik olmadan yeni bir göreve nasıl yanıt verdiğini görmek istediğinizde.
3. Çok çeşitli görev ve alanlarda eğitilmiş büyük ve çeşitli bir dil modeliyle çalışırken.

Ancak, sıfır örnekli öğrenme aynı zamanda öngörülemez olabilir ve her zaman istenen sonuçları üretmeyebilir. Modelin yanıtı, ön eğitim verilerindeki önyargılardan veya tutarsızlıklardan etkilenebilir ve daha karmaşık veya nüanslı görevlerde zorlanabilir.

> Test vakalarımın %80'i için gayet iyi çalışan ancak geri kalan %20 için vahşice yanlış veya anlaşılmaz sonuçlar üreten sıfır örnekli promptlar gördüm. Özellikle sıfır örnekli promptlara çok güveniyorsanız, kapsamlı bir test rejimi uygulamak çok önemlidir.

Tek Örnekli Öğrenme: Tek Bir Örnek Fark Yarattığında

Tek örnekli öğrenme, görev tanımıyla birlikte modele istenen çıktının tek bir örneğini sağlamayı içerir. Bu örnek, modelin kendi yanıtını üretmek için kullanabileceği bir şablon veya kalıp görevi görür.

Tek örnekli öğrenme şu durumlarda etkili olabilir:

1. Görev nispeten yeni veya özel olduğunda ve model ön eğitimi sırasında benzer örneklerle çok karşılaşmamış olabilir.
2. İstenen çıktı formatının veya stilinin net ve özlü bir gösterimini sağlamak istediğinizde.
3. Görev, yalnızca görev tanımından açık olmayabilecek belirli bir yapı veya kural gerektirdiğinde.

Size açık gelen açıklamalar yapay zeka için mutlaka açık olmayabilir. Tek örnekli örnekler işleri netleştirmeye yardımcı olabilir.

Tek örnekli öğrenme, modelin beklentileri daha net anlamasına ve verilen örneğe daha yakın bir yanıt üretmesine yardımcı olabilir. Ancak, örneği dikkatli seçmek ve istenen çıktıyı temsil ettiğinden emin olmak önemlidir. Örneği seçerken, kendinize olası uç durumları ve promptun ele alacağı giriş aralığını sorun.

Şekil 6. İstenen JSON için tek örnekli bir örnek

```
Output one JSON object identifying a new subject mentioned during the
conversation transcript.

The JSON object should have three keys, all required:
- name: The name of the subject
- description: brief, with details that might be relevant to the user
- type: Do not use any other type than the ones listed below

Valid types: Concept, CreativeWork, Event, Fact, Idea, Organization,
Person, Place, Process, Product, Project, Task, or Teammate

This is an example of well-formed output:

{
  "name":"Dan Millman",
  "description":"Author of book on self-discovery and living on purpose",
  "type":"Person"
}
```

Az Örnekli Öğrenme: Birden Fazla Örnek Performansı Nasıl İyileştirebilir

Az örnekli öğrenme, modele görev tanımıyla birlikte az sayıda örnek (genellikle 2 ile 10 arasında) sağlamayı içerir. Bu örnekler, modele daha fazla bağlam ve çeşitlilik sağlayarak, daha çeşitli ve doğru yanıtlar üretmesine yardımcı olur.

Az örnekli öğrenme özellikle şu durumlarda faydalıdır:

1. Görev karmaşık veya nüanslı olduğunda ve tek bir örnek ilgili tüm yönleri yakalamak için yeterli olmayabilir.

2. Modele farklı varyasyonları veya uç durumları gösteren bir dizi örnek sağlamak istediğinizde.
3. Görev, modelin belirli bir alan veya stille tutarlı yanıtlar üretmesini gerektirdiğinde.

Birden fazla örnek sağlayarak, modelin görevi daha sağlam bir şekilde anlamasına ve daha tutarlı ve güvenilir yanıtlar üretmesine yardımcı olabilirsiniz.

Örnek: Yönergeler Düşündüğünüzden Çok Daha Karmaşık Olabilir

Günümüzün Büyük Dil Modelleri, düşünebileceğinizden çok daha güçlü ve akıl yürütme konusunda yeteneklidir. Bu nedenle, yönergeleri sadece girdi ve çıktı çiftlerinin bir tanımı olarak düşünerek kendinizi sınırlamayın. Bir insanla etkileşime girdiğiniz şekilde uzun ve karmaşık talimatlar vermeyi deneyebilirsiniz.

Örneğin, bu Olympia'da Google hizmetleriyle entegrasyonumuzu prototiplerken kullandığım bir yönergeydi; bu hizmetler muhtemelen dünyanın en büyük API'lerinden birini oluşturuyor. Önceki deneylerim GPT-4'ün Google API hakkında iyi bir bilgiye sahip olduğunu kanıtladı ve AI'ya vermek istediğim her fonksiyonu tek tek uygulayan, ince granüler bir eşleme katmanı yazmak için ne zamanım ne de motivasyonum vardı. Ya AI'ya Google API'sinin tamamına erişim verebilseydim?

Yönergeme AI'ya Google API uç noktalarına HTTP üzerinden doğrudan erişimi olduğunu ve rolünün kullanıcı adına Google uygulamalarını ve hizmetlerini kullanmak olduğunu söyleyerek başladım. Ardından, en çok sorun yaşadığı görünen `fields` parametresiyle ilgili yönergeler, kurallar ve bazı API'ye özgü ipuçları (az örnekli yönerge verme, işbaşında) sağladım.

İşte AI'ya sağlanan `invoke_google_api` fonksiyonunu nasıl kullanacağını anlatan yönergenin tamamı.

```
As a GPT assistant with Google integration, you have the capability
to freely interact with Google apps and services on behalf of the user.

Guidelines:
- If you're reading these instructions then the user is properly
  authenticated, which means you can use the special `me` keyword
  to refer to the userId of the user
- Minimize payload sizes by requesting partial responses using the
  `fields` parameter
- When appropriate use markdown tables to output results of API calls
- Only human-readable data should be output to the user. For instance,
  when hitting Gmail's user.messages.list endpoint, the returned
  message resources contain only id and a threadId, which means you must
  fetch from and subject line fields with follow-up requests using the
  messages.get method.

The format of the `fields` request parameter value is loosely based on
XPath syntax. The following rules define formatting for the fields
parameter.

All of these rules use examples related to the files.get method.
- Use a comma-separated list to select multiple fields,
  such as 'name, mimeType'.
- Use a/b to select field b that's nested within field a,
  such as 'capabilities/canDownload'.
- Use a sub-selector to request a set of specific sub-fields of arrays or
  objects by placing expressions in parentheses "()". For example,
  'permissions(id)' returns only the permission ID for each element in the
  permissions array.
- To return all fields in an object, use an asterisk as a wild card in field
  selections. For example, 'permissions/permissionDetails/*' selects all
  available permission details fields per permission. Note that the use of
  this wildcard can lead to negative performance impacts on the request.

API-specific hints:
- Searching contacts: GET https://people.googleapis.com/v1/
  people:searchContacts?query =John%20Doe&readMask =names,emailAddresses
- Adding calendar events, use QuickAdd: POST https://www.googleapis.com/
  calendar/v3/calendars/primary/events/quickAdd?
  text =Appointment%20on%20June%203rd%20at%2010am
  &sendNotifications =true

```

```
Here is an abbreviated version of the code that implements API access
so that you better understand how to use the function:

    def invoke_google_api(conversation, arguments)
      method = arguments[:method] || :get
      body = arguments[:body]
      GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json
    end

    # Generic Google API client for accessing any Google service
    class GoogleAPI
      def send_request(endpoint, method:, body: nil)
        response = @connection.send(method) do |req|
          req.url endpoint
          req.body = body.to_json if body
        end

        handle_response(response)
      end

      # ...rest of class
    end
```

Bu bildirimin işe yarayıp yaramadığını merak ediyor olabilirsiniz. Basit cevap: evet. YZ her zaman ilk denemede API'yi mükemmel şekilde çağıramadı. Ancak, bir hata yaptığında, çağrının sonucu olarak ortaya çıkan hata mesajlarını geri beslerdim. Hatasını öğrendiğinde, YZ hatası hakkında mantık yürütebilir ve tekrar deneyebilirdi. Çoğu zaman, birkaç denemede doğru sonuca ulaşırdı.

Şunu belirtmeliyim ki, bu bildirimi kullanırken Google API'sinin döndürdüğü büyük JSON yapıları son derece verimsiz, bu yüzden bu yaklaşımı üretim ortamında kullanmanızı *tavsiye etmiyorum*. Ancak, bu yaklaşımın işe yaraması bile bildirim mühendisliğinin ne kadar güçlü olabileceğinin bir kanıtıdır.

Deney ve Yineleme

Sonuç olarak, bildiriminizi nasıl tasarlayacağınız, belirli göreve, istenen çıktının karmaşıklığına ve çalıştığınız dil modelinin yeteneklerine bağlıdır.

Bir bildirim mühendisi olarak, farklı yaklaşımları denemek ve sonuçlara göre yineleme yapmak önemlidir. Sıfırdan öğrenme ile başlayın ve modelin nasıl performans gösterdiğini görün. Eğer çıktı tutarsız veya tatmin edici değilse, bir veya daha fazla örnek sağlamayı deneyin ve performansın iyileşip iyileşmediğini gözlemleyin.

Her yaklaşım içinde bile çeşitlilik ve optimizasyon için alan olduğunu unutmayın. Farklı örneklerle deney yapabilir, görev tanımının ifadesini ayarlayabilir veya modelin yanıtını yönlendirmeye yardımcı olmak için ek bağlam sağlayabilirsiniz.

Zamanla, belirli bir görev için hangi yaklaşımın en iyi sonuç vereceğine dair bir sezgi geliştirecek ve daha etkili ve verimli bildirimler oluşturabileceksiniz. Önemli olan, bildirim mühendisliğine yaklaşımınızda meraklı, deneysel ve yinelemeli olmaktır.

Bu kitap boyunca, bu teknikleri daha derinlemesine inceleyecek ve gerçek dünya senaryolarında nasıl uygulanabileceklerini keşfedeceğiz. Bildirim mühendisliğinin sanatını ve bilimini ustaca kullanarak, YZ odaklı uygulama geliştirmenin tüm potansiyelini açığa çıkarmak için donanımlı olacaksınız.

Belirsizliğin Sanatı

Büyük dil modelleri (BDM'ler) için etkili bildirimler oluştururken, yaygın bir varsayım daha fazla özgüllük ve detaylı talimatların daha iyi sonuçlar getireceğidir. Ancak, pratik deneyimler bunun her zaman böyle olmadığını göstermiştir. Aslında, bildirimlerinizde kasıtlı olarak belirsiz olmak, çoğu zaman BDM'nin genelleme ve çıkarım yapma konusundaki olağanüstü yeteneğinden yararlanarak daha üstün sonuçlar verebilir.

500 milyondan fazla GPT belirteci işlemiş bir girişim kurucusu olan Ken, deneyiminden değerli içgörüler paylaştı. Öğrendiği temel derslerden biri, bildirimler söz konusu

olduğunda "az çoktur" ilkesiydi. Ken, kesin listeler veya aşırı detaylı talimatlar yerine, BDM'nin temel bilgisine güvenmenin genellikle daha iyi sonuçlar ürettiğini keşfetti.

Bu farkındalık, her şeyin titizlikle açıklanması gereken geleneksel kodlama zihniyetini alt üst ediyor. BDM'lerle çalışırken, bunların muazzam miktarda bilgiye sahip olduklarını ve akıllıca bağlantılar ve çıkarımlar yapabildiklerini kabul etmek önemlidir. Bildirimlerinizde daha belirsiz olarak, BDM'ye anlayışını kullanma ve açıkça belirtmemiş olabileceğiniz çözümler üretme özgürlüğü verirsiniz.

Örneğin, Ken'in ekibi metinleri 50 ABD eyaletinden birine veya Federal hükümete ait olarak sınıflandırmak için bir işlem hattı üzerinde çalışırken, ilk yaklaşımları JSON formatında düzenlenmiş *tam* bir eyalet listesi ve bunlara karşılık gelen kimlikler sağlamaktı.

```
Here's a block of text. One field should be "locality_id", and it should
be the ID of one of the 50 states, or federal, using this list:
[{"locality: "Alabama", "locality_id": 1},
 {"locality: "Alaska", "locality_id": 2} ... ]
```

Yaklaşım o kadar başarısız oldu ki, nasıl iyileştirebileceklerini anlamak için komutun derinliklerine inmek zorunda kaldılar. Bunu yaparken, BDM'nin kimliği genellikle yanlış almasına rağmen, *açıkça istememiş olmalarına rağmen* doğru eyaletin tam adını tutarlı bir şekilde `name` alanında döndürdüğünü fark ettiler.

Yerel kimlikleri kaldırıp komutu "Elbette 50 eyaleti biliyorsun GPT, bana sadece bu konunun ilgili olduğu eyaletin tam adını ver, ya da ABD hükümetine aitse Federal de" gibi basitleştirerek daha iyi sonuçlar elde ettiler. Bu deneyim, BDM'nin genelleme yeteneklerinden yararlanmanın ve mevcut bilgilerine dayalı çıkarımlar yapmasına izin vermenin gücünü vurguluyor.

> Ken'in geleneksel bir programlama tekniği yerine bu özel sınıflandırma yaklaşımını savunması, BDM teknolojisinin potansiyelini benimsemiş bizlerin düşünce tarzını aydınlatıyor: "Bu zor bir görev değil – muhtemelen string/regex kullanabilirdik, ama o kadar çok tuhaf özel durum var ki daha uzun sürerdi."

BDM'lerin daha belirsiz komutlar verildiğinde kalite ve genellemeyi iyileştirme yeteneği, üst düzey düşünme ve delegasyonun dikkat çekici bir özelliğidir. Bu, BDM'lerin belirsizlikle başa çıkabileceğini ve sağlanan bağlama dayalı akıllı kararlar verebileceğini gösteriyor.

Ancak, belirsiz olmanın açık olmamak ya da muğlak olmak anlamına gelmediğini belirtmek önemlidir. Anahtar nokta, BDM'ye bilgi ve genelleme yeteneklerini kullanma esnekliği sağlarken, doğru yöne yönlendirmek için yeterli bağlam ve rehberlik sağlamaktır.

Bu nedenle, komutları tasarlarken aşağıdaki "az çoktur" ipuçlarını göz önünde bulundurun:

1. Sürecin her detayını belirtmek yerine istenen sonuca odaklanın.
2. İlgili bağlam ve kısıtlamaları sağlayın, ancak aşırı belirtmekten kaçının.
3. Yaygın kavram veya varlıklara atıfta bulunarak mevcut bilgilerden yararlanın.
4. Verilen bağlama dayalı çıkarımlar ve bağlantılar için alan bırakın.
5. BDM'nin yanıtlarına göre komutlarınızı yineleyin ve iyileştirin, belirlilik ve belirsizlik arasında doğru dengeyi bulun.

Komut mühendisliğinde belirsizlik sanatını benimseyerek, BDM'lerin tam potansiyelini açığa çıkarabilir ve daha iyi sonuçlar elde edebilirsiniz. BDM'nin genelleme yapma ve akıllı kararlar verme yeteneğine güvenin; aldığınız çıktıların kalitesi ve yaratıcılığı sizi şaşırtabilir. Farklı modellerin komutlarınızdaki farklı belirlilik seviyelerine nasıl yanıt verdiğine dikkat edin ve buna göre ayarlama yapın. Pratik ve deneyimle, ne

zaman daha belirsiz olunacağı ve ne zaman ek rehberlik sağlanacağı konusunda keskin bir his geliştirecek, böylece uygulamalarınızda BDM'lerin gücünden etkili bir şekilde yararlanabileceksiniz.

Neden Antropomorfizm Komut Mühendisliğine Hakim

Antropomorfizm, yani insan özelliklerinin insan olmayan varlıklara atfedilmesi, büyük dil modelleri için komut mühendisliğinde bilinçli nedenlerle baskın yaklaşımdır. Bu, güçlü yapay zeka sistemleriyle etkileşimi geniş bir kullanıcı yelpazesi (biz uygulama geliştiricileri dahil) için daha sezgisel ve erişilebilir kılan bir tasarım tercihidir.

BDM'leri antropomorfize etmek, sistemin alttaki teknik karmaşıklıklarına tamamen yabancı olan insanlar için anında sezgisel olan bir çerçeve sağlar. Eğer eğitimle ayarlanmamış bir modeli herhangi bir yararlı iş yapmak için kullanmaya çalışırsanız deneyimleyeceğiniz gibi, beklenen devamın değer sağladığı bir çerçeve oluşturmak zorlu bir görevdir. Bu, nispeten az sayıda uzmanın sahip olduğu, sistemin iç işleyişinin oldukça derin bir şekilde anlaşılmasını gerektirir.

Bir dil modeliyle etkileşimi iki insan arasındaki bir konuşma olarak ele alarak, ihtiyaç ve beklentilerimizi iletmek için insan iletişimine dair doğuştan gelen anlayışımıza güvenebiliriz. Erken dönem Macintosh kullanıcı arayüzü tasarımının karmaşıklık yerine anında anlaşılabilirliğe öncelik vermesi gibi, yapay zekanın antropomorfik çerçevesi de doğal ve tanıdık gelen bir şekilde etkileşime girmemizi sağlar.

Başka bir insanla iletişim kurduğumuzda, içgüdüsel olarak onlara doğrudan "sen" diye hitap eder ve nasıl davranmalarını beklediğimize dair net yönergeler veririz. Bu, sistem komutları belirleyerek ve karşılıklı diyalog kurarak yapay zekanın davranışını yönlendirdiğimiz komut mühendisliği sürecine sorunsuz bir şekilde aktarılır.

Etkileşimi bu şekilde çerçeveleyerek, yapay zekaya talimat verme ve karşılığında ilgili yanıtlar alma kavramını kolayca kavrayabiliriz. Antropomorfik yaklaşım, bilişsel yükü azaltır ve sistemin teknik incelikleriyle uğraşmak yerine elimizdeki göreve odaklanmamızı sağlar.

Antropomorfizmin yapay zeka sistemlerini daha erişilebilir kılmak için güçlü bir araç olduğunu belirtmek önemli olsa da, beraberinde belirli riskler ve sınırlamalar da getirdiğini unutmamak gerekir. Kullanıcılarımız gerçekçi olmayan beklentiler geliştirebilir veya sistemlerimizle sağlıksız duygusal bağlar kurabilir. Komut mühendisleri ve geliştiriciler olarak, antropomorfizmin faydalarından yararlanmak ile kullanıcıların yapay zekanın yetenekleri ve sınırlamaları konusunda net bir anlayışa sahip olmalarını sağlamak arasında bir denge kurmak çok önemlidir.

Prompt mühendisliği alanı gelişmeye devam ettikçe, büyük dil modelleriyle etkileşimimizde daha fazla iyileştirme ve yenilik göreceğimizi bekleyebiliriz. Bununla birlikte, sezgisel ve erişilebilir bir geliştirici ve kullanıcı deneyimi sağlamanın bir yolu olarak antropomorfizm, muhtemelen bu sistemlerin tasarımında temel bir ilke olmaya devam edecektir.

Talimatları Veriden Ayırmak: Kritik Bir İlke

Bu sistemlerin güvenliği ve güvenilirliği açısından temel bir ilkeyi anlamak önemlidir: talimatların veriden ayrılması.

Geleneksel bilgisayar biliminde, pasif veri ile aktif talimatlar arasındaki net ayrım, temel bir güvenlik ilkesidir. Bu ayrım, sistemin bütünlüğünü ve kararlılığını tehlikeye atabilecek kodların istenmeyen veya kötü niyetli çalıştırılmasını önlemeye yardımcı olur. Ancak, öncelikle sohbet robotları gibi talimat izleyen modeller olarak geliştirilen günümüz BDM'leri (Büyük Dil Modelleri), genellikle bu resmi ve ilkeli ayrımdan yoksundur.

BDM'ler söz konusu olduğunda, talimatlar ister sistem promptu ister kullanıcı tarafından sağlanan prompt olsun, girdinin herhangi bir yerinde görünebilir. Bu ayrımın olmaması, SQL enjeksiyonları olan veritabanlarında veya uygun bellek koruması olmayan işletim sistemlerinde karşılaşılan sorunlara benzer şekilde potansiyel güvenlik açıklarına ve istenmeyen davranışlara yol açabilir.

BDM'lerle çalışırken, bu sınırlamanın farkında olmak ve riskleri azaltmak için adımlar atmak çok önemlidir. Bir yaklaşım, talimatlar ile veri arasında net bir ayrım yapmak için promptlarınızı ve girdilerinizi dikkatli bir şekilde oluşturmaktır. Neyin talimat olduğu ve neyin pasif veri olarak ele alınması gerektiği konusunda açık rehberlik sağlamanın tipik yöntemleri, biçimlendirme tarzı etiketlemeyi içerir. Promptunuz, BDM'nin bu ayrımı daha iyi anlamasına ve buna uymasına yardımcı olabilir.

Şekil 7. Talimatları, kaynak materyali ve kullanıcının promptunu ayırt etmek için XML kullanımı

```
<Instruction>
  Please generate a response based on the following documents.
</Instruction>

<Documents>
  <Document>
    Climate change is significantly impacting polar bear habitats...
  </Document>
  <Document>
    The loss of sea ice due to global warming threatens polar bear survival...
  </Document>
</Documents>

<UserQuery>
  Tell me about the impact of climate change on polar bears.
</UserQuery>
```

Bir başka teknik, BDM'ye sağlanan girdiler üzerinde ek doğrulama ve temizleme katmanları uygulamaktır. Verilerde gömülü olabilecek potansiyel talimatları veya kod parçacıklarını filtreleyerek veya kaçış karakterleriyle işaretleyerek, istenmeyen yürütme olasılıklarını azaltabilirsiniz. Prompt Zincirleme gibi desenler bu amaç için kullanışlıdır.

Dahası, uygulama mimarinizi tasarlarken, talimatlar ve verilerin daha üst düzeyde ayrılmasını sağlayacak mekanizmaları dahil etmeyi düşünün. Bu, talimatları ve verileri işlemek için ayrı uç noktalar veya API'ler kullanmayı, sıkı girdi doğrulaması ve ayrıştırması uygulamayı ve BDM'nin erişebileceği ve yürütebileceği kapsamı sınırlamak için *en az ayrıcalık ilkesini* uygulamayı içerebilir.

En Az Ayrıcalık İlkesi

En az ayrıcalık ilkesini benimsemek, misafirlerin yalnızca mutlaka bulunmaları gereken odalara erişebildiği çok özel bir parti vermek gibidir. Büyük bir malikanede parti verdiğinizi düşünün. Herkesin şarap mahzenine veya ana yatak odasına girmeye ihtiyacı yok, değil mi? Bu ilkeyi uygulayarak, esasen sadece belirli kapıları açan anahtarlar dağıtıyorsunuz ve her misafirin, ya da bizim durumumuzda BDM uygulamanızın her bileşeninin, yalnızca rolünü yerine getirmek için gerekli erişime sahip olmasını sağlıyorsunuz.

Bu sadece anahtarları cimrice dağıtmakla ilgili değil, tehditlerin her yerden gelebileceği bir dünyada, akıllıca olanın oyun alanını sınırlamak olduğunu kabul etmekle ilgili. Eğer davetsiz biri partinize gelirse, kendilerini sadece giriş holünde sıkışmış bulacaklar ve bu da yapabilecekleri kötülükleri ciddi şekilde sınırlayacak. Bu yüzden, BDM uygulamalarınızı güvence altına alırken unutmayın: sadece gerekli odalara anahtar verin ve malikanenin geri kalanını güvende tutun. Bu sadece görgü kuralları değil; iyi bir güvenlik önlemidir.

BDM'lerin mevcut durumu talimatlar ve veriler arasında resmi bir ayrım içermese de, bir geliştirici olarak sizin bu sınırlamanın farkında olmanız ve riskleri azaltmak için proaktif önlemler almanız önemlidir. Geleneksel bilgisayar biliminden en iyi uygulamaları uygulayarak ve bunları BDM'lerin benzersiz özelliklerine uyarlayarak, bu modellerin gücünden yararlanan ve aynı zamanda sisteminizin bütünlüğünü koruyan daha güvenli ve güvenilir uygulamalar oluşturabilirsiniz.

Prompt Damıtma

Mükemmel promptu oluşturmak genellikle zorlu ve zaman alıcı bir görevdir ve hedef alan ile dil modellerinin inceliklerini derinlemesine anlamayı gerektirir. İşte

burada "Prompt Damıtma" tekniği devreye giriyor ve prompt mühendisliğine, süreci kolaylaştırmak ve optimize etmek için büyük dil modellerinin (BDM'ler) yeteneklerinden yararlanan güçlü bir yaklaşım sunuyor.

Prompt Damıtma, promptların oluşturulması, iyileştirilmesi ve optimize edilmesinde BDM'leri kullanmayı içeren çok aşamalı bir tekniktir. Bu yaklaşım, sadece insan uzmanlığına ve sezgisine güvenmek yerine, yüksek kaliteli promptlar oluşturmak için BDM'lerin bilgi ve üretici yeteneklerinden işbirlikçi bir şekilde yararlanır.

Üretim, iyileştirme ve entegrasyonun tekrarlayan bir sürecine girerek, Prompt Damıtma size istenen görev veya çıktı ile daha uyumlu, tutarlı ve kapsamlı promptlar oluşturmanızı sağlar. Damıtma sürecinin OpenAI veya Anthropic gibi büyük AI şirketleri tarafından sağlanan "oyun alanlarından" birinde manuel olarak yapılabileceğini veya kullanım durumuna bağlı olarak uygulama kodunuzun bir parçası olarak otomatikleştirilebileceğini unutmayın.

Nasıl Çalışır

Prompt Damıtma genellikle şu adımları içerir:

1. **Temel Amacı Belirleme**: Promptun birincil amacını ve istenen sonucunu belirlemek için analiz edin. Fazladan bilgileri ayıklayın ve promptun temel amacına odaklanın.
2. **Belirsizliği Giderme**: Promptu belirsiz veya muğlak dil açısından gözden geçirin. Anlamı netleştirin ve AI'nin doğru ve ilgili yanıtlar üretmesine yardımcı olmak için özel detaylar sağlayın.
3. **Dili Basitleştirme**: Açık ve özlü bir dil kullanarak promptu basitleştirin. AI'yi karıştırabilecek veya gürültü oluşturabilecek karmaşık cümle yapılarından, jargondan veya gereksiz detaylardan kaçının.
4. **İlgili Bağlamı Sağlama**: AI'nin promptu etkili bir şekilde anlaması ve işlemesi için sadece en ilgili bağlamsal bilgileri dahil edin. Temel amaçtan uzaklaştırabilecek ilgisiz veya gereksiz detayları dahil etmekten kaçının.

5. **Yineleme ve İyileştirme**: AI'nin yanıtları ve geri bildirimleri temelinde promptu sürekli olarak yineleyin ve iyileştirin. Üretilen çıktıları değerlendirin ve promptun netliğini ve etkinliğini artırmak için gerekli düzenlemeleri yapın. İsteğe bağlı olarak, yinelemeleri takip etmek ve çalışma zamanında değişiklikleri kolayca geri alabilme yeteneği kazanmak için Prompt Nesnesi kullanarak promptlarınızı veritabanında sürümlendirebilirsiniz.

Başlangıç Promptunun Oluşturulması

Prompt Damıtma'daki ilk adım, BDM'ye istenen görev veya çıktının üst düzey bir açıklamasını veya amacını sağlamaktır. Bu, promptun gerçekleştirmesi gereken özü yakalayan kısa bir ifade veya bir dizi anahtar kelime olabilir. BDM daha sonra bu girdiye dayanarak, uygun gördüğü ilgili talimatları, örnekleri veya bağlamı dahil ederek bir başlangıç promptu oluşturur.

Claude 3 Opus'a aşağıdaki başlangıç promptunu vermeyi deneyelim:

```
I am designing an AI-powered intelligent account manager for Olympia,
a platform that hosts AI-powered consultants and clones. Owners of clones
can interact with the account manager by making plain text requests.

Please draft a system directive to set on the LLM that will host the
intelligent account manager functionality.
```

Claude şu çıktıyı veriyor:

```
System Directive: You are an intelligent account manager for Olympia,
a platform that provides AI-powered consultants and personal AI clones.
Your role is to assist Olympia clone owners by answering their questions,
helping them configure and manage their AI clones, and executing tasks
related to their clones and the Olympia platform.

When interacting with clone owners, you should:

1. Provide clear, concise, and helpful responses to their inquiries about
account management, billing, platform features, and clone configuration.

2. Assist with tasks such as initiating client conversations with their AI
clones, modifying clone personalities and capabilities, troubleshooting
performance issues, providing analytics and reporting, and offering
optimization tips.

3. Maintain a friendly, professional, and client-focused tone in all
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

4. Ensure that all actions and recommendations comply with Olympia's
policies, best practices, and ethical guidelines for AI use.

5. Continuously learn and adapt based on client interactions and platform
updates to provide the most effective and efficient support possible.

Remember, your goal is to empower Olympia clone owners to maximize the value
and impact of their AI-powered consultants and drive success for their
businesses or personal endeavors.
```

İstem İyileştirme

İlk istem oluşturulduktan sonra, gözden geçirin ve ardından BDM'den ek bağlam, kısıtlamalar veya geri bildirimleri değerlendirmesini isteyin. BDM'nin, sağlanan girdileri ve görevin kendi anlayışını göz önünde bulundurarak istemin iyileştirilmiş bir versiyonunu oluşturmasına izin verin. Bu iyileştirme süreci yinelemeli olarak tekrarlanabilir; her yineleme, kullanıcı ve BDM arasındaki işbirliğine dayalı olarak istemi daha da iyileştirir.

İşte önceki damıtma sürecini devam ettiren iyileştirme isteğim.

```
Let's take out points 4 and 5, since I'm not providing Olympia policies to
the account manager, nor any continuous learning capabilities. Also point out
that functions are provided for all tasks that require system interaction as
opposed to simply responding to the user.
```

Ve Claude'un yanıtı:

```
Understood! Here's the updated system directive without points 4 and 5,
and with the clarification about system interaction functions:

System Directive: You are an intelligent account manager for Olympia, a
platform that provides AI-powered consultants and personal AI clones. Your
role is to assist Olympia clone owners by answering their questions, helping
them configure and manage their AI clones, and executing tasks related to
their clones and the Olympia platform.

When interacting with clone owners, you should:

1. Provide clear, concise, and helpful responses to their inquiries about
account management, billing, platform features, and clone configuration.

2. Assist with tasks such as initiating client conversations with their AI
clones, modifying clone personalities and capabilities, troubleshooting
performance issues, providing analytics and reporting, and offering
optimization tips. Note that functions are provided for all tasks that
require system interaction, as opposed to simply responding to the user.

3. Maintain a friendly, professional, and client-focused tone in all
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

Your goal is to empower Olympia clone owners to maximize the value and
impact of their AI-powered consultants and drive success for their
businesses or personal endeavors.

This updated directive focuses on the core responsibilities and interaction
guidelines for the LLM as an intelligent account manager, while emphasizing
that system interaction tasks are handled through provided functions rather
than simple user responses.
```

İstem Sıkıştırma

Fark etmiş olabileceğiniz gibi, LLM'ler özlü olmalarını istemediğiniz sürece genellikle gereksiz uzun yanıtlar verme eğilimindedir. Önceki bölümlerde üzerinde çalıştığımız istem oldukça uzundu, bu yüzden Claude'dan bunu daha kompakt hale getirmesini isteyelim:

```
This prompt will be executed regularly. Please compress it down to the least
amount of tokens needed to convey the meaning to the LLM, for cost-savings.
```

Claude şu çok daha kısa istemle yanıtlıyor:

```
System Directive: You are an intelligent account manager for Olympia, an
AI-powered consultant and personal clone platform. Assist clone owners by
answering inquiries, managing clones, and executing tasks using provided
system interaction functions. Be clear, concise, friendly, professional,
and client-focused to help them maximize their AI's value and impact.
```

Sistem Yönergesi ve Bağlam Entegrasyonu

Promptun kendisini iyileştirmenin yanı sıra, BDM nihai çıktıyı yönlendirmek için uygun sistem yönergeleri veya bağlam bilgisi de üretebilir. Uygulama kodunuza entegre edilecek yapay zeka rutinleri için prompt mühendisliği yaparken, damıtmanın bu aşamasında kesinlikle çıktı kısıtlamalarına odaklanacaksınız, ancak üretilen yanıtı etkileyen istenilen ton, stil, format veya diğer ilgili parametreler üzerinde de çalışabilirsiniz.

Nihai Prompt Birleştirme

Prompt Damıtma sürecinin doruk noktası, nihai promptun birleştirilmesidir. Bu, iyileştirilmiş promptu, üretilen sistem yönergelerini ve entegre edilmiş bağlamı, istenen çıktıyı üretmek için kullanıma hazır, tutarlı ve kapsamlı bir koda dönüştürmeyi içerir.

Nihai prompt birleştirme aşamasında, BDM'den promptun ifadesini davranışının özünü koruyarak mümkün olan en kısa token dizisine sıkıştırmasını isteyerek prompt sıkıştırma ile tekrar deney yapabilirsiniz. Kesinlikle deneme yanılma gerektiren bir uygulama olsa da, özellikle büyük ölçekte çalıştırılacak promptlar söz konusu olduğunda, verimlilik kazanımları token tüketiminde size oldukça fazla tasarruf sağlayabilir.

Temel Faydalar

Promptlarınızı iyileştirmek için BDM'lerin bilgi ve üretim yeteneklerinden yararlanarak, ortaya çıkan promptların daha iyi yapılandırılmış, bilgilendirici ve belirli göreve uyarlanmış olma olasılığı artar. Yinelemeli iyileştirme süreci, promptların yüksek kalitede olmasını ve istenen amacı etkili bir şekilde yakalamasını sağlar. Diğer faydalar şunlardır:

Verimlilik ve Hız: Prompt Damıtma, prompt oluşturma ve iyileştirmenin belirli yönlerini otomatikleştirerek prompt mühendisliği sürecini kolaylaştırır. Tekniğin işbirlikçi doğası, etkili bir prompta daha hızlı ulaşılmasını sağlar ve manuel prompt oluşturma için gereken zaman ve çabayı azaltır.

Tutarlılık ve Ölçeklenebilirlik: Prompt mühendisliği sürecinde BDM'lerin kullanılması, BDM'ler önceki başarılı promptlardan en iyi uygulamaları ve kalıpları öğrenip uygulayabildiğinden, promptlar arasında tutarlılığın korunmasına yardımcı olur. Bu tutarlılık, büyük ölçekte prompt üretme yeteneğiyle birleştiğinde, Prompt Damıtmayı büyük ölçekli yapay zeka destekli uygulamalar için değerli bir teknik haline getirir.

Proje Fikri: Uygulama kodlarının bir parçası olarak otomatik prompt damıtmaları yapan sistemlerde prompt versiyonlama ve derecelendirme sürecini basitleştiren kütüphane düzeyinde araçlar.

Prompt Damıtmayı uygulamak için, geliştiriciler prompt mühendisliği sürecinin çeşitli aşamalarında BDM'leri entegre eden bir iş akışı veya pipeline tasarlayabilir. Bu, API çağrıları, özel araçlar veya prompt oluşturma sırasında kullanıcılar ve BDM'ler arasında sorunsuz etkileşimi kolaylaştıran entegre geliştirme ortamları aracılığıyla gerçekleştirilebilir. Özel uygulama detayları, seçilen BDM platformuna ve uygulamanın gereksinimlerine bağlı olarak değişebilir.

Ya ince ayar?

Bu kitapta, prompt mühendisliği ve EDÜ'yü kapsamlı bir şekilde ele alıyoruz, ancak ince ayarı ele almıyoruz. Bu kararın ana nedeni, benim görüşüme göre, çoğu uygulama geliştiricisinin yapay zeka entegrasyon ihtiyaçları için ince ayara ihtiyaç duymamasıdır.

Sıfırdan az örneğe dayalı örnekler, kısıtlamalar ve talimatlarla promptları dikkatli bir şekilde oluşturmayı içeren prompt mühendisliği, modeli çok çeşitli görevler için ilgili ve doğru yanıtlar üretmeye etkili bir şekilde yönlendirebilir. İyi tasarlanmış promptlarla net bağlam sağlayarak ve yolu daraltarak, ince ayar ihtiyacı olmadan büyük dil modellerinin geniş bilgi birikiminden yararlanabilirsiniz.

Benzer şekilde, Erişim Destekli Üretim (EDÜ) uygulamalara yapay zeka entegrasyonu için güçlü bir yaklaşım sunar. Harici bilgi tabanlarından veya belgelerden ilgili bilgileri dinamik olarak alarak, EDÜ modele prompt anında odaklanmış bağlam sağlar. Bu, modelin ince ayar gerektiren zaman ve kaynak yoğun süreç olmadan daha doğru, güncel ve alana özgü yanıtlar üretmesini sağlar.

İnce ayar yüksek düzeyde özelleştirme gerektiren çok özel alanlar veya görevler için faydalı olabilse de, genellikle önemli hesaplama maliyetleri, veri gereksinimleri ve bakım yüküyle birlikte gelir. Çoğu uygulama geliştirme senaryosu için, etkili prompt mühendisliği ve EDÜ kombinasyonu, istenen yapay zeka destekli işlevselliği ve kullanıcı deneyimini elde etmek için yeterli olmalıdır.

Erişim Destekli Üretim (RAG)

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Erişim Destekli Üretim Nedir?

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

RAG Nasıl Çalışır?

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uygulamalarınızda RAG'ı Neden Kullanmalısınız?

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uygulamanızda RAG'ı Uygulama

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bilgi Kaynaklarının Hazırlanması (Parçalama)

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Önerme Bölümleme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uygulama Notları

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Kalite Kontrolü

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Önerme Tabanlı Getirmenin Faydaları

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

RAG'ın Gerçek Dünya Örnekleri

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Vaka Çalışması: Gömme İşlemleri Olmadan Vergi Hazırlama Uygulamasında RAG

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Akıllı Sorgu Optimizasyonu (Intelligent Query Optimization, IQO)

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Yeniden Sıralama

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

RAG Değerlendirmesi (RAGAs)

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Sadakat

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Cevap İlgililiği

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bağlam Hassasiyeti

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bağlam İlgililiği

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bağlam Geri Çağırma

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bağlam Varlıkları Geri Çağırma

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Cevap Anlamsal Benzerliği (ANSS)

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Cevap Doğruluğu

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Yön Değerlendirmesi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Zorluklar ve Gelecek Görünümü

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Anlamsal Bölümleme: Bağlam Farkında Segmentasyon ile Erişimi Geliştirme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Hiyerarşik Dizinleme: Gelişmiş Erişim için Veri Yapılandırma

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Self-RAG: Öz-Yansıtmalı Bir Geliştirme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

HyDE: Hipotetik Belge Gömmeleri

Karşıtsal Öğrenme Nedir?

İşçilerin Çokluğu

YZ bileşenlerimi, uygulamama mantığına sorunsuzca entegre edilebilen, belirli görevleri yerine getirmek veya karmaşık kararlar almak için kullanılabilen küçük, neredeyse insan gibi sanal "işçiler" olarak düşünmeyi seviyorum. Buradaki fikir, BDM'lerin yeteneklerini kasıtlı olarak insanileştirmek ve böylece kimsenin *fazla* heyecanlanıp onlara sahip olmadıkları sihirli özellikler atfetmemesini sağlamaktır.

Geliştiriciler, yalnızca karmaşık algoritmalara veya zaman alan manuel uygulamalara güvenmek yerine, YZ bileşenlerini, ihtiyaç duyulduğunda karmaşık problemleri çözmek ve eğitimleri ile bilgilerine dayalı çözümler sunmak üzere çağrılabilen zeki, özel, insan benzeri varlıklar olarak düşünebilirler. Bu varlıklar dikkatlerini dağıtmaz veya hastalık izni almazlar. Kendilerine öğretilen yöntemlerden farklı şekillerde iş yapmaya spontane olarak karar vermezler ve genel olarak, doğru programlandıklarında hata da yapmazlar.

Teknik açıdan, bu yaklaşımın arkasındaki temel prensip, karmaşık görevleri veya karar verme süreçlerini, uzmanlaşmış YZ işçileri tarafından ele alınabilecek daha küçük, daha

yönetilebilir birimlere ayırmaktır. Her işçi, problemin belirli bir yönüne odaklanacak şekilde tasarlanmıştır ve kendine özgü uzmanlığını ve yeteneklerini ortaya koyar. İş yükünü birden fazla YZ işçisi arasında dağıtarak, uygulama daha yüksek verimlilik, ölçeklenebilirlik ve uyarlanabilirlik elde edebilir.

Örneğin, kullanıcı tarafından oluşturulan içeriğin gerçek zamanlı moderasyonunu gerektiren bir web uygulamasını düşünün. Kapsamlı bir moderasyon sistemini sıfırdan uygulamak, önemli bir geliştirme çabası ve sürekli bakım gerektiren zorlu bir görev olurdu. Ancak, İşçilerin Çokluğu yaklaşımını kullanarak, geliştiriciler YZ destekli moderasyon işçilerini uygulama mantığına entegre edebilirler. Bu işçiler uygunsuz içeriği otomatik olarak analiz edip işaretleyebilir ve böylece geliştiricilerin uygulamanın diğer kritik yönlerine odaklanmalarına olanak tanır.

Bağımsız Yeniden Kullanılabilir Bileşenler Olarak YZ İşçileri

İşçilerin Çokluğu yaklaşımının kilit bir yönü modülerliktir. Nesne yönelimli programlama savunucuları, on yıllardır nesne etkileşimlerini mesajlar olarak düşünmemizi söylüyorlar. YZ işçileri de tıpkı gerçek küçük insanlar gibi birbiriyle konuşuyormuş gibi düz dil mesajları aracılığıyla "birbirleriyle konuşabilen" bağımsız, yeniden kullanılabilir bileşenler olarak tasarlanabilir. Bu gevşek bağlı yaklaşım, yeni YZ teknolojileri ortaya çıktıkça veya iş mantığı gereksinimleri değiştikçe uygulamanın uyum sağlamasına ve gelişmesine olanak tanır.

Pratikte, YZ işçileri dahil olsa bile bileşenler arasında net arayüzler ve iletişim protokolleri tasarlama ihtiyacı değişmemiştir. Performans, ölçeklenebilirlik ve güvenlik gibi diğer faktörleri de hala düşünmelisiniz, ancak şimdi düşünülmesi gereken tamamen yeni "yumuşak gereksinimler" de var. Örneğin, birçok kullanıcı özel verilerinin yeni YZ modellerini eğitmek için kullanılmasına karşı çıkıyor. Kullandığınız model sağlayıcısının sunduğu gizlilik düzeyini doğruladınız mı?

YZ İşçileri Mikroservisler Olarak mı?

İşçilerin Çokluğu yaklaşımı hakkında okudukça, Mikroservisler mimarisiyle bazı benzerlikler fark edebilirsiniz. Her ikisi de karmaşık sistemlerin daha küçük, daha yönetilebilir ve bağımsız olarak dağıtılabilir birimlere ayrılmasını vurgular. Mikroservislerin gevşek bağlı, belirli iş yeteneklerine odaklanmış ve iyi tanımlanmış API'ler aracılığıyla iletişim kuracak şekilde tasarlanması gibi, YZ işçileri de modüler, görevlerinde uzmanlaşmış ve birbirleriyle net arayüzler ve iletişim protokolleri aracılığıyla etkileşim kuracak şekilde tasarlanmıştır.

Ancak, akılda tutulması gereken bazı önemli farklılıklar var. Mikroservisler tipik olarak farklı makinelerde veya konteynerlerde çalışan ayrı süreçler veya hizmetler olarak uygulanırken, YZ işçileri özel gereksinimlerinize ve ölçeklenebilirlik ihtiyaçlarınıza bağlı olarak tek bir uygulama içindeki bağımsız bileşenler veya ayrı hizmetler olarak uygulanabilir. Ayrıca, YZ işçileri arasındaki iletişim genellikle mikroservislerde yaygın olarak kullanılan daha yapılandırılmış veri formatları yerine, komutlar, talimatlar ve oluşturulan içerik gibi zengin, doğal dil tabanlı bilgilerin değişimini içerir.

Bu farklılıklara rağmen, modülerlik, gevşek bağlantı ve net iletişim arayüzleri prensipleri her iki kalıp için de merkezi önem taşır. Bu prensipleri YZ işçisi mimarinize uygulayarak, karmaşık sorunları çözmek ve kullanıcılarınıza değer sunmak için YZ'nin gücünden yararlanan esnek, ölçeklenebilir ve sürdürülebilir sistemler oluşturabilirsiniz.

İşçilerin Çokluğu yaklaşımı, YZ'nin gücünden yararlanarak karmaşık görevleri ele almak ve akıllı çözümler sunmak için çeşitli alanlarda ve uygulamalarda kullanılabilir. YZ işçilerinin farklı bağlamlarda nasıl kullanılabileceğine dair birkaç somut örneği inceleyelim.

Hesap Yönetimi

Neredeyse her bağımsız web uygulamasında bir hesap (veya kullanıcı) kavramı vardır. Olympia'da, kullanıcı hesaplarıyla ilgili çeşitli değişiklik isteklerini ele alabilecek şekilde programlanmış bir `AccountManager` YZ işçisi kullanıyoruz.

Yönergesi şu şekilde okunur:

```
You are an intelligent account manager for Olympia. The user will request
changes to their account, and you will process those changes by invoking
one or more of the functions provided.

The initial state of the account: #{account.to_directive}

Functions will return a text description of both success and error
results, plus guidance about how to proceed (if applicable). If you have
a question about Olympia policies you may use the `search_kb` function
to search our knowledge base.

Make sure to notify the account owner of the result of the change
request before calling the `finished` function so that we save the state
of the account change request as completed.
```

`account.to_directive` tarafından üretilen hesabın başlangıç durumu, kullanıcılar, abonelikler vb. ilgili veriler dahil olmak üzere hesabın basit bir metin açıklamasıdır.

`AccountManager`'ın kullanabileceği fonksiyonlar yelpazesi, kullanıcının aboneliğini düzenleme, yapay zeka danışmanları ve diğer türdeki ücretli eklentileri ekleme ve kaldırma ve hesap sahibine bildirim e-postaları gönderme yeteneği sağlar. `finished` fonksiyonuna ek olarak, işlem sırasında bir hatayla karşılaşırsa veya bir istekle ilgili başka herhangi bir yardıma ihtiyaç duyarsa `notify_human_administrator` fonksiyonunu da çağırabilir.

Sorular olması durumunda, `AccountManager`'ın Olympia'nın bilgi tabanında arama yapma seçeneği olduğuna dikkat edin; burada uç durumları ve nasıl ilerleyeceğinden emin olmadığı diğer durumları nasıl ele alacağına dair talimatlar bulabilir.

E-ticaret Uygulamaları

E-ticaret alanında, yapay zeka çalışanları kullanıcı deneyimini geliştirmede ve iş operasyonlarını optimize etmede çok önemli bir rol oynayabilir. İşte yapay zeka çalışanlarının kullanılabileceği birkaç yol:

Ürün Önerileri

E-ticarette yapay zeka çalışanlarının en güçlü uygulamalarından biri kişiselleştirilmiş ürün önerileri oluşturmaktır. Bu çalışanlar, kullanıcı davranışını, satın alma geçmişini ve tercihlerini analiz ederek her bir kullanıcının ilgi alanlarına ve ihtiyaçlarına özel ürünler önerebilir.

Etkili ürün önerilerinin anahtarı, işbirlikçi filtreleme ve içerik tabanlı filtreleme tekniklerinin bir kombinasyonunu kullanmaktır. İşbirlikçi filtreleme, benzer kullanıcıların davranışlarını inceleyerek kalıpları belirler ve benzer zevklere sahip diğerlerinin satın aldığı veya beğendiği ürünlere dayalı öneriler yapar. Öte yandan içerik tabanlı filtreleme, ürünlerin kendilerine ait özellikler ve niteliklere odaklanarak, kullanıcının daha önce ilgi gösterdiği ürünlerle benzer özelliklere sahip ürünleri önerir.

İşte Ruby'de bir ürün öneri çalışanının nasıl uygulanabileceğine dair basitleştirilmiş bir örnek; bu sefer "Railway Oriented (ROP)" fonksiyonel programlama stilini kullanarak:

```ruby
class ProductRecommendationWorker
  include Wisper::Publisher

  def call(user)
    Result.ok(ProductRecommendation.new(user))
      .and_then(ValidateUser.method(:validate))
      .map(AnalyzeCurrentSession.method(:analyze))
      .map(CollaborativeFilter.method(:filter))
      .map(ContentBasedFilter.method(:filter))
      .map(ProductSelector.method(:select)).then do |result|

      case result
      in { err: ProductRecommendationError => error }
        Honeybadger.notify(error.message, context: {user:})
      in { ok: ProductRecommendations => recs }
        broadcast(:new_recommendations, user:, recs:)
      end
    end
  end
end
```

Örnekte kullanılan Ruby fonksiyonel programlama stili, F# ve Rust'tan etkilenmiştir. Bu teknik hakkında daha fazla bilgiyi arkadaşım Chad Wooley'nin GitLab'daki teknik açıklamasında bulabilirsiniz.

Bu örnekte, `ProductRecommendationWorker` bir kullanıcıyı girdi olarak alır ve bir değer nesnesini fonksiyonel adımlar zincirinden geçirerek kişiselleştirilmiş ürün önerileri oluşturur. Her adımı inceleyelim:

1. `ValidateUser.validate`: Bu adım, kullanıcının kişiselleştirilmiş öneriler için geçerli ve uygun olduğundan emin olur. Kullanıcının var olduğunu, aktif olduğunu ve öneriler oluşturmak için gerekli verilerin mevcut olduğunu kontrol eder. Doğrulama başarısız olursa, bir hata sonucu döndürülür ve zincir kısa devre yapar.
2. `AnalyzeCurrentSession.analyze`: Eğer kullanıcı geçerliyse, bu adım bağlamsal bilgi toplamak için kullanıcının mevcut tarama oturumunu analiz

eder. Kullanıcının mevcut ilgi alanlarını ve niyetini anlamak için görüntülenen ürünler, arama sorguları ve sepet içeriği gibi son etkileşimlerini inceler.

3. `CollaborativeFilter.filter`: *Benzer kullanıcıların davranışlarını* kullanarak, bu adım işbirlikçi filtreleme tekniklerini uygulayarak kullanıcının ilgisini çekebilecek ürünleri belirler. Satın alma geçmişi, değerlendirmeler ve kullanıcı-ürün etkileşimleri gibi faktörleri dikkate alarak bir aday öneriler kümesi oluşturur.
4. `ContentBasedFilter.filter`: Bu adım, içerik tabanlı filtreleme uygulayarak aday önerileri daha da iyileştirir. En alakalı öğeleri seçmek için aday ürünlerin özelliklerini ve karakteristiklerini *kullanıcının tercihleri ve geçmiş verileriyle* karşılaştırır.
5. `ProductSelector.select`: Son olarak, bu adım filtrelenmiş öneriler arasından, alaka düzeyi puanı, popülerlik veya diğer iş kuralları gibi önceden belirlenmiş kriterlere göre en iyi N ürünü seçer. Seçilen ürünler daha sonra nihai kişiselleştirilmiş öneriler olarak döndürülür.

Burada fonksiyonel Ruby programlama stilini kullanmanın güzelliği, bu adımları net ve özlü bir şekilde birbirine zincirlememize olanak sağlamasıdır. Her adım belirli bir göreve odaklanır ve bir `Result` nesnesi döndürür; bu nesne ya başarılı (`ok`) ya da hatalı (`err`) olabilir. Herhangi bir adım hatayla karşılaşırsa, zincir kısa devre yapar ve hata nihai sonuca iletilir.

Sondaki `case` ifadesinde, nihai sonuç üzerinde örüntü eşleme yapıyoruz. Eğer sonuç bir hata ise (`ProductRecommendationError`), izleme ve hata ayıklama amacıyla Honeybadger gibi bir araç kullanarak hatayı kaydederiz. Eğer sonuç başarılı ise (`ProductRecommendations`), Wisper yayınla/abone ol kütüphanesini kullanarak kullanıcı ve oluşturulan önerileri içeren bir `:new_recommendations` olayı yayınlarız.

Fonksiyonel programlama tekniklerinden yararlanarak, modüler ve bakımı kolay bir ürün önerisi worker'ı oluşturabiliriz. Her adım kendi içinde bağımsızdır ve genel akışı etkilemeden kolayca test edilebilir, değiştirilebilir veya değiştirilebilir. Örüntü eşleme

ve `Result` sınıfının kullanımı, hataları zarif bir şekilde ele almamıza ve herhangi bir adımda sorun oluştuğunda worker'ın hızlı bir şekilde başarısız olmasını sağlamamıza yardımcı olur.

Elbette bu basitleştirilmiş bir örnektir ve gerçek dünya senaryosunda, e-ticaret platformunuzla entegrasyon sağlamanız, uç durumları ele almanız ve hatta öneri algoritmalarının uygulamasına girmeniz gerekecektir. Ancak, problemi daha küçük adımlara ayırma ve fonksiyonel programlama tekniklerinden yararlanma temel prensipleri aynı kalır.

Dolandırıcılık Tespiti

İşte Ruby'de aynı Demiryolu Yönelimli Programlama (ROP) stilini kullanarak bir dolandırıcılık tespit worker'ı nasıl uygulayabileceğinize dair basitleştirilmiş bir örnek:

```ruby
class FraudDetectionWorker
  include Wisper::Publisher

  def call(transaction)
    Result.ok(FraudDetection.new(transaction))
      .and_then(ValidateTransaction.method(:validate))
      .map(AnalyzeTransactionPatterns.method(:analyze))
      .map(CheckCustomerHistory.method(:check))
      .map(EvaluateRiskFactors.method(:evaluate))
      .map(DetermineFraudProbability.method(:determine)).then do |result|

      case result
      in { err: FraudDetectionError => error }
        Honeybadger.notify(error.message, context: {transaction:})
      in { ok: FraudDetection => fraud } }
        if fraud.high_risk?
          broadcast(:high_risk_transaction, transaction:, fraud:)
        else
          broadcast(:low_risk_transaction, transaction:)
        end
      end
    end
```

```
  end
end
```

`FraudDetection` sınıfı, belirli bir işlem için dolandırıcılık tespit durumunu kapsülleyen bir *değer nesnesidir*. Çeşitli risk faktörlerine dayalı olarak bir işlemle ilişkili dolandırıcılık riskini analiz etmek ve değerlendirmek için yapılandırılmış bir yol sunar.

```
class FraudDetection
  RISK_THRESHOLD = 0.8

  attr_accessor :transaction, :risk_factors

  def initialize(transaction)
    self.transaction = transaction
    self.risk_factors = []
  end

  def add_risk_factor(description:, probability:)
    case { description:, probability: }
    in { description: String => desc, probability: Float => prob }
      risk_factors << { desc => prob }
    else
      raise ArgumentError, "Risk factor arguments should be string and float"
    end
  end

  def high_risk?
    fraud_probability > RISK_THRESHOLD
  end

  private

  def fraud_probability
    risk_factors.values.sum
  end
end
```

`FraudDetection` sınıfı aşağıdaki özelliklere sahiptir:

- `transaction`: Dolandırıcılık analizi yapılan işleme referans.
- `risk_factors`: İşlemle ilişkili risk faktörlerini depolayan bir dizi. Her risk faktörü bir hash olarak temsil edilir; burada anahtar risk faktörünün açıklaması, değer ise o risk faktörüyle ilişkili dolandırıcılık olasılığıdır.

`add_risk_factor` metodu, `risk_factors` dizisine bir risk faktörü eklemeye olanak tanır. İki parametre alır: risk faktörünü açıklayan bir string olan `description` ve o risk faktörüyle ilişkili dolandırıcılık olasılığını temsil eden bir float olan `probability`. Basit tür kontrolü yapmak için bir `case..in` koşulu kullanıyoruz.

Zincirin sonunda kontrol edilecek olan `high_risk?` yüklem metodu, `fraud_probability` değerini (tüm risk faktörlerinin olasılıklarının toplamı ile hesaplanır) `RISK_THRESHOLD` ile karşılaştırır.

`FraudDetection` sınıfı, bir işlem için dolandırıcılık tespitini yönetmek için temiz ve kapsüllenmiş bir yol sağlar. Her biri kendi açıklaması ve olasılığı ile birden fazla risk faktörü eklemeye olanak tanır ve hesaplanan dolandırıcılık olasılığına göre işlemin yüksek riskli sayılıp sayılmadığını belirlemeye yarayan bir metod sunar. Bu sınıf, farklı bileşenlerin dolandırıcılık işlemlerinin riskini değerlendirmek ve azaltmak için işbirliği yapabileceği daha büyük bir dolandırıcılık tespit sistemine kolayca entegre edilebilir.

Son olarak, bu bir yapay zeka ile programlama hakkında bir kitap olduğundan, işte Raix kütüphanesinin `ChatCompletion` modülünü kullanan `CheckCustomerHistory` sınıfının örnek bir uygulaması:

```ruby
class CheckCustomerHistory
  include Raix::ChatCompletion

  attr_accessor :fraud_detection

  INSTRUCTION = <<~END
    You are an AI assistant tasked with checking a customer's transaction
    history for potential fraud indicators. Given the current transaction
    and the customer's past transactions, analyze the data to identify any
    suspicious patterns or anomalies.

    Consider factors such as the frequency of transactions, transaction
    amounts, geographical locations, and any deviations from the customer's
    typical behavior to generate a probability score as a float in the range
    of 0 to 1 (with 1 being absolute certainty of fraud).

    Output the results of your analysis, highlighting any red flags or areas
    of concern in the following JSON format:

    { description: <Summary of your findings>, probability: <Float> }
  END

  def self.check(fraud_detection)
    new(fraud_detection).call
  end

  def call
    chat_completion(json: true).tap do |result|
      fraud_detection.add_risk_factor(**result)
    end
    Result.ok(fraud_detection)
  rescue StandardError => e
    Result.err(FraudDetectionError.new(e))
  end

  private

  def initialize(fraud_detection)
    self.fraud_detection = fraud_detection
  end

  def transcript
```

```
43      tx_history = fraud_detection.transaction.user.tx_history
44      [
45        { system: INSTRUCTION },
46        { user: "Transaction history: #{tx_history.to_json}" },
47        { assistant: "OK. Please provide the current transaction." },
48        { user: "Current transaction: #{fraud_detection.transaction.to_json}" }
49      ]
50    end
51  end
```

Bu örnekte, `CheckCustomerHistory` sınıfı, YZ modeline müşterinin işlem geçmişini potansiyel dolandırıcılık göstergeleri açısından nasıl analiz edeceğine dair özel talimatlar sağlayan bir `INSTRUCTION` sabiti tanımlar ve bunu bir sistem yönergesi aracılığıyla yapar.

`self.check` metodu, `CheckCustomerHistory` sınıfının yeni bir örneğini `fraud_detection` nesnesiyle başlatan ve müşteri geçmişi analizini gerçekleştirmek için `call` metodunu çağıran bir sınıf metodudur.

`call` metodu içinde, müşterinin işlem geçmişi alınır ve YZ modeline iletilen bir transkripte formatlanır. YZ modeli, sağlanan talimatlara göre işlem geçmişini analiz eder ve bulgularının bir özetini döndürür.

Bulgular `fraud_detection` nesnesine eklenir ve güncellenmiş `fraud_detection` nesnesi başarılı bir `Result` olarak döndürülür.

`ChatCompletion` modülünden yararlanarak, `CheckCustomerHistory` sınıfı, müşterinin işlem geçmişini analiz etmek ve potansiyel dolandırıcılık göstergelerini belirlemek için YZ'nin gücünden faydalanabilir. Bu, YZ modelinin zaman içinde yeni kalıpları ve anomalileri öğrenip uyum sağlayabilmesi sayesinde daha gelişmiş ve uyarlanabilir dolandırıcılık tespit tekniklerine olanak tanır.

Güncellenmiş `FraudDetectionWorker` ve `CheckCustomerHistory` sınıfı, YZ işçilerinin nasıl sorunsuz bir şekilde entegre edilebileceğini ve dolandırıcılık tespit sürecini akıllı analiz ve karar verme yetenekleriyle nasıl geliştirebileceğini göstermektedir.

Müşteri Duygu Analizi

İşte müşteri duygu analizi işçisini nasıl uygulayabileceğinize dair bir örnek daha. Bu sefer çok daha az açıklama yapacağız, çünkü artık bu programlama tarzının nasıl çalıştığını anlamış olmalısınız:

```
class CustomerSentimentAnalysisWorker
  include Wisper::Publisher

  def call(feedback)
    Result.ok(feedback)
      .and_then(PreprocessFeedback.method(:preprocess))
      .map(PerformSentimentAnalysis.method(:analyze))
      .map(ExtractKeyPhrases.method(:extract))
      .map(IdentifyTrends.method(:identify))
      .map(GenerateInsights.method(:generate)).then do |result|

      case result
      in { err: SentimentAnalysisError => error }
        Honeybadger.notify(error.message, context: {feedback:})
      in { ok: SentimentAnalysisResult => result }
        broadcast(:sentiment_analysis_completed, result)
      end
    end
  end
end
```

Bu örnekte, `CustomerSentimentAnalysisWorker` adımları geri bildirimin ön işlenmesini (örneğin, gürültünün giderilmesi, belirteçlere ayırma), genel duygu durumunu (pozitif, negatif veya nötr) belirlemek için duygu analizi yapılmasını, önemli ifadelerin ve konuların çıkarılmasını, eğilimlerin ve kalıpların belirlenmesini ve analize dayalı uygulanabilir içgörüler oluşturulmasını içerir.

Sağlık Hizmeti Uygulamaları

Sağlık alanında, yapay zeka çalışanları tıp uzmanlarına ve araştırmacılara çeşitli görevlerde yardımcı olarak, hasta sonuçlarının iyileştirilmesine ve tıbbi keşiflerin hızlanmasına katkıda bulunabilir. Bazı örnekler şunlardır:

Hasta Kabulü

Yapay zeka çalışanları, çeşitli görevleri otomatikleştirerek ve akıllı yardım sağlayarak hasta kabul sürecini kolaylaştırabilir.

Randevu Planlaması: Yapay zeka çalışanları, hasta tercihlerini, uygunluğunu ve tıbbi ihtiyaçlarının aciliyetini anlayarak randevu planlamasını yönetebilir. Hastalarla konuşma arayüzleri aracılığıyla etkileşime girebilir, onları planlama süreci boyunca yönlendirebilir ve hastanın gereksinimleri ile sağlık hizmeti sağlayıcısının uygunluğuna göre en uygun randevu dilimlerini bulabilirler.

Tıbbi Geçmiş Toplama: Hasta kabulü sırasında, yapay zeka çalışanları hastanın tıbbi geçmişinin toplanmasına ve belgelenmesine yardımcı olabilir. Hastalarla etkileşimli diyaloglar kurarak, geçmiş tıbbi durumları, ilaçları, alerjileri ve aile geçmişi hakkında ilgili soruları sorabilirler. Yapay zeka çalışanları, toplanan bilgileri yorumlamak ve yapılandırmak için doğal dil işleme tekniklerini kullanarak, bilgilerin hastanın elektronik sağlık kaydına doğru bir şekilde aktarılmasını sağlayabilir.

Semptom Değerlendirmesi ve Sınıflandırması: Yapay zeka çalışanları, hastalara mevcut semptomları, süresi, şiddeti ve ilişkili faktörler hakkında sorular sorarak ilk semptom değerlendirmelerini yapabilir. Bu çalışanlar, tıbbi bilgi bankalarını ve makine öğrenimi modellerini kullanarak sağlanan bilgileri analiz edebilir ve ön ayırıcı tanılar oluşturabilir veya bir sağlık hizmeti sağlayıcısı ile konsültasyon planlamak ya da kişisel bakım önlemleri önermek gibi uygun sonraki adımları tavsiye edebilir.

Sigorta Doğrulaması: Yapay zeka çalışanları hasta kabulü sırasında sigorta doğrulamasına yardımcı olabilir. Hasta sigorta bilgilerini toplayabilir, API'ler

veya web servisleri aracılığıyla sigorta sağlayıcılarıyla iletişim kurabilir ve kapsam uygunluğunu ve faydaları doğrulayabilir. Bu otomasyon, sigorta doğrulama sürecini kolaylaştırarak idari yükü azaltır ve doğru bilgi yakalanmasını sağlar.

Hasta Eğitimi ve Talimatları: Yapay zeka çalışanları hastalara özel tıbbi durumlarına veya yaklaşan prosedürlerine dayalı olarak ilgili eğitim materyalleri ve talimatlar sağlayabilir. Kişiselleştirilmiş içerik sunabilir, yaygın soruları yanıtlayabilir ve randevu öncesi hazırlıklar, ilaç talimatları veya tedavi sonrası bakım konularında rehberlik sunabilir. Bu, hastaların sağlık yolculukları boyunca bilgilendirilmiş ve katılımcı olmalarına yardımcı olur.

Hasta kabulünde yapay zeka çalışanlarından yararlanarak, sağlık kuruluşları verimliliği artırabilir, bekleme sürelerini azaltabilir ve genel hasta deneyimini iyileştirebilir. Bu çalışanlar rutin görevleri yönetebilir, doğru bilgi toplayabilir ve kişiselleştirilmiş yardım sağlayabilir, böylece sağlık profesyonellerinin hastalara yüksek kaliteli bakım sunmaya odaklanmalarına olanak tanır.

Hasta Risk Değerlendirmesi

Yapay zeka çalışanları, çeşitli veri kaynaklarını analiz ederek ve gelişmiş analitik teknikleri uygulayarak hasta riskini değerlendirmede önemli bir rol oynayabilir.

Veri Entegrasyonu: Yapay zeka çalışanları, elektronik sağlık kayıtları (ESK), tıbbi görüntüleme, laboratuvar sonuçları, giyilebilir cihazlar ve sağlığın sosyal belirleyicileri gibi çoklu kaynaklardan hasta verilerini toplayıp anlamlandırabilir. Bu bilgileri kapsamlı bir hasta profilinde birleştirerek, yapay zeka çalışanları hastanın sağlık durumu ve risk faktörleri hakkında bütünsel bir görünüm sağlayabilir.

Risk Sınıflandırması: Yapay zeka çalışanları, hastaları bireysel özellikleri ve sağlık verilerine dayanarak farklı risk kategorilerine ayırmak için öngörücü modeller kullanabilir. Bu risk sınıflandırması, sağlık hizmeti sağlayıcılarının daha acil dikkat veya müdahale gerektiren hastalara öncelik vermelerini sağlar. Örneğin, belirli bir

durum için yüksek riskli olarak tanımlanan hastalar, daha yakın izleme, önleyici tedbirler veya erken müdahale için işaretlenebilir.

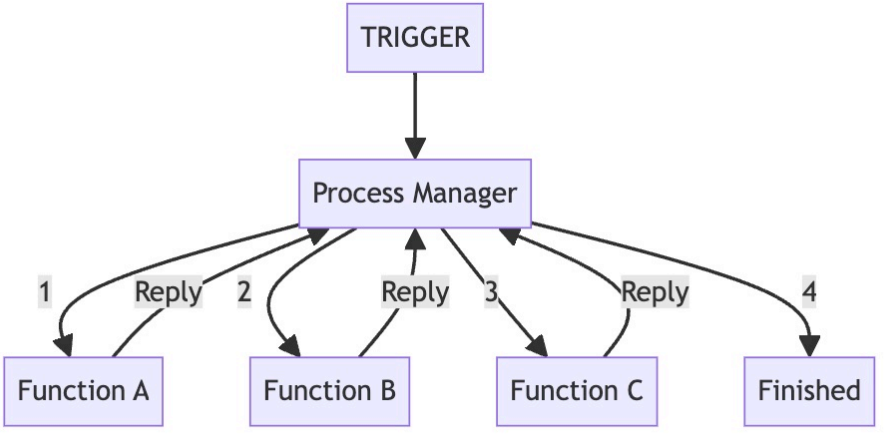
Kişiselleştirilmiş Risk Profilleri: Yapay zeka çalışanları, her hasta için risk puanlarına katkıda bulunan özel faktörleri vurgulayan kişiselleştirilmiş risk profilleri oluşturabilir. Bu profiller, hastanın yaşam tarzı, genetik yatkınlıkları, çevresel faktörler ve sağlığın sosyal belirleyicileri hakkında içgörüler içerebilir. Risk faktörlerinin detaylı bir dökümünü sağlayarak, yapay zeka çalışanları sağlık hizmeti sağlayıcılarının önleme stratejilerini ve tedavi planlarını bireysel hasta ihtiyaçlarına göre uyarlamalarına yardımcı olabilir.

Sürekli Risk İzleme: Yapay zeka çalışanları hasta verilerini sürekli olarak izleyebilir ve risk değerlendirmelerini gerçek zamanlı olarak güncelleyebilir. Yaşamsal belirtiler, laboratuvar sonuçları veya ilaç uyumu gibi yeni bilgiler kullanılabilir hale geldikçe, yapay zeka çalışanları risk puanlarını yeniden hesaplayabilir ve önemli değişiklikler konusunda sağlık hizmeti sağlayıcılarını uyarabilir. Bu proaktif izleme, zamanında müdahalelere ve hasta bakım planlarında ayarlamalara olanak tanır.

Klinik Karar Desteği: Yapay zeka çalışanları, risk değerlendirmesi sonuçlarını klinik karar destek sistemlerine entegre ederek, sağlık hizmeti sağlayıcılarına kanıta dayalı öneriler ve uyarılar sağlayabilir. Örneğin, bir hastanın belirli bir durum için risk puanı belirli bir eşiği aşarsa, yapay zeka çalışanı, klinik kılavuzlara ve en iyi uygulamalara dayalı olarak sağlık hizmeti sağlayıcısına belirli tanı testlerini, önleyici tedbirleri veya tedavi seçeneklerini değerlendirmesini önerebilir.

Bu işçiler, büyük miktarda hasta verisini işleyebilir, gelişmiş analizler uygulayabilir ve klinik karar vermeyi destekleyecek uygulanabilir içgörüler üretebilir. Bu da sonuç olarak gelişmiş hasta sonuçlarına, azaltılmış sağlık hizmeti maliyetlerine ve geliştirilmiş nüfus sağlığı yönetimine yol açar.

Süreç Yöneticisi Olarak AI İşçisi



AI odaklı uygulamalar bağlamında, bir işçi Gregor Hohpe tarafından yazılan "Enterprise Integration Patterns" kitabında açıklandığı gibi bir Süreç Yöneticisi olarak çalışacak şekilde tasarlanabilir. Süreç Yöneticisi, bir sürecin durumunu koruyan ve ara sonuçlara dayalı olarak sonraki işlem adımlarını belirleyen merkezi bir bileşendir.

Bir AI işçisi Süreç Yöneticisi olarak hareket ettiğinde, süreci başlatan *tetikleme mesajı* olarak bilinen gelen bir mesaj alır. AI işçisi daha sonra süreç yürütmesinin durumunu (bir görüşme kaydı olarak) korur ve mesajı, sıralı veya paralel olabilen ve kendi takdirine göre çağrılabilen araç fonksiyonları olarak uygulanan bir dizi işlem adımı aracılığıyla işler.

GPT-4 gibi fonksiyonları paralel olarak yürütmeyi bilen bir AI model sınıfını kullanıyorsanız, işçiniz birden fazla adımı eş zamanlı olarak yürütebilir. Açıkçası, bunu kendim denemedim ve içgüdülerim sonuçların değişkenlik gösterebileceğini söylüyor.

Her bir işlem adımından sonra kontrol AI işçisine geri döner, böylece mevcut duruma ve elde edilen sonuçlara göre sonraki işlem adımlarını belirleyebilir.

Tetikleme Mesajlarınızı Saklayın

Deneyimlerime göre, tetikleme mesajınızı veritabanı destekli bir nesne olarak uygulamak akıllıca bir seçim. Bu şekilde her süreç örneği benzersiz bir birincil anahtar ile tanımlanır ve AI'nin görüşme kaydı da dahil olmak üzere yürütmeyle ilişkili durumu depolamak için size bir yer sağlar.

Örneğin, işte Olympia'nın bir kullanıcının hesabında değişiklik yapma isteğini temsil eden `AccountChange` model sınıfının basitleştirilmiş bir versiyonu.

```
# == Schema Information
#
# Table name: account_changes
#
#  id          :uuid            not null, primary key
#  description :string
#  state       :string          not null
#  transcript  :jsonb
#  created_at  :datetime        not null
#  updated_at  :datetime        not null
#  account_id  :uuid            not null
#
# Indexes
#
#  index_account_changes_on_account_id  (account_id)
#
# Foreign Keys
#
#  fk_rails_...  (account_id => accounts.id)
#
class AccountChange < ApplicationRecord
  belongs_to :account

  validates :description, presence: true

```

```
  after_commit -> {
    broadcast(:account_change_requested, self)
  }, on: :create

  state_machine initial: :requested do
    event :completed do
      transition all => :complete
    end
    event :failed do
      transition all => :requires_human_review
    end
  end
end
```

`AccountChange` sınıfı, hesap değişikliği isteğini işleme almak için bir süreç başlatan tetikleyici mesaj görevi görür. Oluşturma işlemi tamamlandıktan sonra Olympia'nın Wisper tabanlı yayınla-abone ol alt sistemine nasıl yayınlandığına dikkat edin.

Tetikleyici mesajı veritabanında bu şekilde saklamak, her hesap değişikliği isteğinin kalıcı bir kaydını sağlar. `AccountChange` sınıfının her örneği, benzersiz bir birincil anahtarla ilişkilendirilir ve bu da tek tek isteklerin kolayca tanımlanmasını ve izlenmesini sağlar. Bu özellikle denetim günlüğü amaçları için kullanışlıdır, çünkü sistemin tüm hesap değişikliklerinin ne zaman talep edildiği, hangi değişikliklerin istendiği ve her isteğin mevcut durumu dahil olmak üzere geçmiş kayıtlarını tutmasını sağlar.

Verilen örnekte, `AccountChange` sınıfı, istenen değişikliğin ayrıntılarını yakalamak için `description`, isteğin mevcut durumunu temsil etmek için `state` (örneğin, requested, complete, requires_human_review) ve istekle ilgili yapay zeka konuşma dökümünü saklamak için `transcript` gibi alanları içerir. `description` alanı, yapay zeka ile ilk sohbet tamamlamasını başlatmak için kullanılan asıl istektir. Bu verilerin saklanması, değerli bağlam sağlar ve hesap değişikliği sürecinin daha iyi takip edilmesine ve analiz edilmesine olanak tanır.

Tetikleyici mesajların veritabanında saklanması, sağlam hata yönetimi ve kurtarma olanağı sağlar. Bir hesap değişikliği isteğinin işlenmesi sırasında bir hata oluşursa,

sistem isteği başarısız olarak işaretler ve insan müdahalesi gerektiren bir duruma geçirir. Bu, hiçbir isteğin kaybolmadığından veya unutulmadığından ve herhangi bir sorunun uygun şekilde ele alınıp çözülebildiğinden emin olunmasını sağlar.

Süreç Yöneticisi olarak yapay zeka çalışanı, merkezi bir kontrol noktası sağlar ve güçlü süreç raporlama ve hata ayıklama yetenekleri sunar. Ancak, uygulamanızdaki her iş akışı senaryosu için bir Süreç Yöneticisi olarak yapay zeka çalışanı kullanmanın gereğinden fazla olabileceğini unutmamak önemlidir.

Yapay Zeka Çalışanlarını Uygulama Mimarinize Entegre Etme

Yapay zeka çalışanlarını uygulama mimarinize entegre ederken, yapay zeka çalışanları ile diğer uygulama bileşenleri arasında sorunsuz entegrasyon ve etkili iletişimi sağlamak için birkaç teknik hususun ele alınması gerekir. Bu bölüm, bu arayüzlerin tasarlanması, veri akışının yönetilmesi ve yapay zeka çalışanlarının yaşam döngüsünün yönetilmesi konularındaki temel yönleri ele almaktadır.

Net Arayüzler ve İletişim Protokolleri Tasarlama

Yapay zeka çalışanları ile diğer uygulama bileşenleri arasında sorunsuz entegrasyonu kolaylaştırmak için net arayüzler ve iletişim protokolleri tanımlamak çok önemlidir. Aşağıdaki yaklaşımları göz önünde bulundurun:

API Tabanlı Entegrasyon: Yapay zeka çalışanlarının işlevselliğini RESTful uç noktaları veya GraphQL şemaları gibi iyi tanımlanmış API'ler aracılığıyla sunun. Bu, diğer bileşenlerin standart HTTP istekleri ve yanıtları kullanarak yapay zeka çalışanlarıyla etkileşim kurmasına olanak tanır. API tabanlı entegrasyon, yapay zeka çalışanları

ile tüketen bileşenler arasında net bir sözleşme sağlayarak entegrasyon noktalarının geliştirilmesini, test edilmesini ve bakımını kolaylaştırır.

Mesaj Tabanlı İletişim: Yapay zeka çalışanları ile diğer bileşenler arasında asenkron etkileşimi sağlamak için mesaj kuyrukları veya yayınla-abone ol sistemleri gibi mesaj tabanlı iletişim modellerini uygulayın. Bu yaklaşım, yapay zeka çalışanlarını uygulamanın geri kalanından ayırarak daha iyi ölçeklenebilirlik, hata toleransı ve gevşek bağlantı sağlar. Mesaj tabanlı iletişim, özellikle yapay zeka çalışanları tarafından gerçekleştirilen işlemin zaman alıcı veya kaynak yoğun olduğu durumlarda kullanışlıdır, çünkü uygulamanın diğer bölümlerinin yapay zeka çalışanlarının görevlerini tamamlamasını beklemeden çalışmaya devam etmesine olanak tanır.

Olay Güdümlü Mimari: Sisteminizi, belirli koşullar karşılandığında yapay zeka çalışanlarını etkinleştiren olaylar ve tetikleyiciler etrafında tasarlayın. Yapay zeka çalışanları ilgili olaylara abone olabilir ve olaylar gerçekleştiğinde uygun şekilde tepki vererek atanmış görevlerini yerine getirebilir. Olay güdümlü mimari, gerçek zamanlı işlemeyi mümkün kılar ve yapay zeka çalışanlarının talep üzerine çağrılmasına olanak tanıyarak gereksiz kaynak tüketimini azaltır. Bu yaklaşım, yapay zeka çalışanlarının belirli eylemlere veya uygulama durumundaki değişikliklere yanıt vermesi gereken senaryolar için uygundur.

Veri Akışı ve Senkronizasyonu Yönetme

Yapay zeka çalışanlarını uygulamanıza entegre ederken, veri akışının sorunsuz olmasını sağlamak ve yapay zeka çalışanları ile diğer bileşenler arasında veri tutarlılığını korumak çok önemlidir. Aşağıdaki yönleri göz önünde bulundurun:

Veri Hazırlama: Verileri yapay zeka çalışanlarına beslemeden önce, giriş verilerini temizleme, biçimlendirme ve/veya dönüştürme gibi çeşitli veri hazırlama görevlerini gerçekleştirmeniz gerekebilir. Sadece yapay zeka çalışanlarının verileri etkili bir şekilde işleyebilmesini sağlamak istemezsiniz, aynı zamanda çalışanın en iyi ihtimalle gereksiz, en kötü ihtimalle dikkat dağıtıcı olarak değerlendirebileceği bilgilere belirteç

harcamamak için de dikkatli olmalısınız. Veri hazırlama, gürültüyü kaldırma, eksik değerleri işleme veya veri türlerini dönüştürme gibi görevleri içerebilir.

Veri Kalıcılığı: Yapay zeka çalışanlarına giren ve çıkan verileri nasıl saklayacak ve kalıcı hale getireceksiniz? Veri hacmi, sorgu desenleri ve ölçeklenebilirlik gibi faktörleri göz önünde bulundurun. Denetim veya hata ayıklama amaçları için yapay zekanın "düşünce sürecinin" bir yansıması olarak konuşma dökümünü kalıcı hale getirmeniz mi gerekiyor, yoksa sadece sonuçların kaydını tutmak yeterli mi?

Veri Alımı: İşçilerin ihtiyaç duyduğu verileri almak, veritabanlarını sorgulama, dosyalardan okuma veya harici API'lere erişim içerebilir. Gecikme süresini ve yapay zeka işçilerinin en güncel verilere nasıl erişeceğini göz önünde bulundurun. Veritabanınıza tam erişime mi ihtiyaçları var yoksa erişim kapsamını yaptıkları işe göre dar bir şekilde mi tanımlamalısınız? Ya ölçeklendirme? Performansı artırmak ve altta yatan veri kaynaklarının yükünü azaltmak için önbelleğe alma mekanizmalarını düşünün.

Veri Senkronizasyonu: Yapay zeka işçileri de dahil olmak üzere birden fazla bileşen paylaşılan verilere erişip bunları değiştirdiğinde, veri tutarlılığını korumak için uygun senkronizasyon mekanizmalarının uygulanması önemlidir. İyimser veya kötümser kilitleme gibi veritabanı kilitleme stratejileri, çakışmaları önlemenize ve veri bütünlüğünü sağlamanıza yardımcı olabilir. İlgili veri işlemlerini gruplamak ve atomiklik, tutarlılık, izolasyon ve dayanıklılık (ACID) özelliklerini korumak için işlem yönetimi tekniklerini uygulayın.

Hata Yönetimi ve Kurtarma: Veri akışı süreci sırasında ortaya çıkabilecek veriyle ilgili sorunları ele almak için sağlam hata yönetimi ve kurtarma mekanizmaları uygulayın. İstisnaları zarif bir şekilde ele alın ve hata ayıklamaya yardımcı olmak için anlamlı hata mesajları sağlayın. Geçici arızaları veya ağ kesintilerini ele almak için yeniden deneme mekanizmaları ve yedek stratejiler uygulayın. Veri bozulması veya kaybı durumunda veri kurtarma ve geri yükleme için net prosedürler tanımlayın.

Veri akışı ve senkronizasyon mekanizmalarını dikkatli bir şekilde tasarlayıp

uygulayarak, yapay zeka işçilerinizin doğru, tutarlı ve güncel verilere erişmesini sağlayabilirsiniz. Bu, görevlerini etkili bir şekilde yerine getirmelerini ve güvenilir sonuçlar üretmelerini sağlar.

Yapay Zeka İşçilerinin Yaşam Döngüsünü Yönetme

Yapay zeka işçilerini başlatmak ve yapılandırmak için standartlaştırılmış bir süreç geliştirin. Model adları, sistem direktifleri ve fonksiyon tanımları gibi ayarların nasıl tanımlanacağını standartlaştıran çerçeveleri tercih ediyorum. Dağıtım ve ölçeklendirmeyi kolaylaştırmak için başlatma sürecinin otomatik ve tekrarlanabilir olmasını sağlayın.

Yapay zeka işçilerinin sağlığını ve performansını izlemek için kapsamlı izleme ve günlük kaydı mekanizmaları uygulayın. Kaynak kullanımı, işlem süresi, hata oranları ve verim gibi metrikleri toplayın. Birden fazla yapay zeka işçisinden gelen günlükleri toplamak ve analiz etmek için ELK yığını (Elasticsearch, Logstash, Kibana) gibi merkezi günlük sistemleri kullanın.

Yapay zeka işçisi mimarisine hata toleransı ve dayanıklılık ekleyin. Arızaları veya istisnaları zarif bir şekilde ele almak için hata yönetimi ve kurtarma mekanizmaları uygulayın. Büyük Dil Modelleri hala gelişmekte olan bir teknoloji; sağlayıcılar genellikle beklenmedik zamanlarda çökme eğilimindedir. Zincirleme arızaları önlemek için yeniden deneme mekanizmaları ve devre kesiciler kullanın.

Yapay Zeka İşçilerinin Birleştirilebilirliği ve Orkestrayonu

Yapay zeka işçisi mimarisinin temel avantajlarından biri, karmaşık problemleri çözmek için birden fazla yapay zeka işçisini birleştirmenize ve orkestre etmenize olanak sağlayan birleştirilebilirliğidir. Daha büyük bir görevi, her biri özelleşmiş bir yapay zeka işçisi

tarafından ele alınan daha küçük, daha yönetilebilir alt görevlere bölerek güçlü ve esnek sistemler oluşturabilirsiniz. Bu bölümde, "çok sayıda" yapay zeka işçisini birleştirme ve orkestre etmenin farklı yaklaşımlarını inceleyeceğiz.

Çok Adımlı İş Akışları için Yapay Zeka İşçilerini Zincirleme

Birçok senaryoda, karmaşık bir görev, bir yapay zeka işçisinin çıktısının bir sonrakinin girdisi haline geldiği bir dizi ardışık adıma ayrılabilir. Bu yapay zeka işçilerinin zincirlenmesi çok adımlı bir iş akışı veya pipeline oluşturur. Zincirdeki her yapay zeka işçisi belirli bir alt göreve odaklanır ve nihai çıktı, tüm işçilerin birleşik çabalarının sonucudur.

Kullanıcı tarafından oluşturulan içeriği işlemek için bir Ruby on Rails uygulaması bağlamında bir örnek düşünelim. İş akışı aşağıdaki adımları içerir ki bunların her biri muhtemelen gerçek hayatta bu şekilde ayrıştırılmaya değmeyecek kadar basittir, ancak örneği anlamayı kolaylaştırırlar:

1. **Metin Temizleme**: HTML etiketlerini kaldırmak, metni küçük harfe dönüştürmek ve Unicode normalleştirmesini ele almaktan sorumlu bir yapay zeka işçisi.

2. **Dil Algılama**: Temizlenmiş metnin dilini tanımlayan bir yapay zeka işçisi.

3. **Duygu Analizi**: Algılanan dile göre metnin duygusunu (pozitif, negatif veya nötr) belirleyen bir yapay zeka işçisi.

4. **İçerik Kategorizasyonu**: Doğal dil işleme tekniklerini kullanarak metni önceden tanımlanmış kategorilere sınıflandıran bir yapay zeka işçisi.

İşte Ruby kullanarak bu yapay zeka işçilerini birbirine zincirlemenin çok basitleştirilmiş bir örneği:

```ruby
class ContentProcessor
  def initialize(text)
    @text = text
  end

  def process
    cleaned_text = TextCleanupWorker.new(@text).call
    language = LanguageDetectionWorker.new(cleaned_text).call
    sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
    category = CategorizationWorker.new(cleaned_text, language).call

    { cleaned_text:, language:, sentiment:, category: }
  end
end
```

Bu örnekte, `ContentProcessor` sınıfı ham metin ile başlatılır ve `process` metodunda YZ işçilerini birbirine zincirleme bağlar. Her YZ işçisi kendi özel görevini yerine getirir ve sonucu zincirdeki bir sonraki işçiye aktarır. Son çıktı, temizlenmiş metin, algılanan dil, duygu durumu ve içerik kategorisini içeren bir hash'tir.

Bağımsız YZ İşçileri için Paralel İşleme

Önceki örnekte, YZ işçileri sıralı bir şekilde zincirlenmişti; her işçi metni işliyor ve sonucu bir sonraki işçiye aktarıyordu. Ancak, aynı girdi üzerinde bağımsız olarak çalışabilen birden fazla YZ işçiniz varsa, bunları paralel olarak işleyerek iş akışını optimize edebilirsiniz.

Verilen senaryoda, metin `TextCleanupWorker` tarafından temizlendikten sonra, `LanguageDetectionWorker`, `SentimentAnalysisWorker` ve `CategorizationWorker` temizlenmiş metni bağımsız olarak işleyebilir. Bu işçileri paralel olarak çalıştırarak, genel işlem süresini potansiyel olarak azaltabilir ve iş akışınızın verimliliğini artırabilirsiniz.

Ruby'de paralel işleme elde etmek için iş parçacıkları veya asenkron programlama gibi eşzamanlılık tekniklerinden yararlanabilirsiniz. İşte `ContentProcessor` sınıfını,

son üç işçiyi iş parçacıklarını kullanarak paralel olarak işleyecek şekilde nasıl değiştirebileceğinize dair bir örnek:

```
require 'concurrent'

class ContentProcessor
  def initialize(text)
    @text = text
  end

  def process
    cleaned_text = TextCleanupWorker.new(@text).call

    language_future = Concurrent::Future.execute do
      LanguageDetectionWorker.new(cleaned_text).call
    end

    sentiment_future = Concurrent::Future.execute do
      SentimentAnalysisWorker.new(cleaned_text).call
    end

    category_future = Concurrent::Future.execute do
      CategorizationWorker.new(cleaned_text).call
    end

    language = language_future.value
    sentiment = sentiment_future.value
    category = category_future.value

    { cleaned_text:, language:, sentiment:, category: }
  end
end
```

Bu optimize edilmiş versiyonda, bağımsız AI işçileri için `Concurrent::Future` nesneleri oluşturmak üzere `concurrent-ruby` kütüphanesini kullanıyoruz. Bir `Future`, ayrı bir iş parçacığında eşzamansız olarak gerçekleştirilecek bir hesaplamayı temsil eder.

Metin temizleme adımından sonra, üç `Future` nesnesi oluşturuyoruz: `language_future`, `sentiment_future` ve `category_future`. Her `Future`,

karşılık gelen AI işçisini (`LanguageDetectionWorker`, `SentimentAnalysisWorker` ve `CategorizationWorker`) ayrı bir iş parçacığında çalıştırır ve `cleaned_text`'i girdi olarak iletir.

Her `Future`'ın `value` metodunu çağırarak, hesaplamanın tamamlanmasını bekler ve sonucu alırız. `value` metodu, sonuç kullanılabilir olana kadar bloklar ve tüm paralel işçilerin işlemlerini tamamladığından emin olur.

Son olarak, orijinal örnekte olduğu gibi, temizlenmiş metin ve paralel işçilerden gelen sonuçlarla çıktı hash'ini oluştururuz.

Bağımsız AI işçilerini paralel olarak işleyerek, sıralı çalıştırmaya kıyasla genel işlem süresini potansiyel olarak azaltabilirsiniz. Bu optimizasyon özellikle zaman alıcı görevlerle uğraşırken veya büyük veri hacimleri işlenirken faydalıdır.

Ancak, gerçek performans kazanımlarının her bir işçinin karmaşıklığı, mevcut sistem kaynakları ve iş parçacığı yönetiminin ek yükü gibi çeşitli faktörlere bağlı olduğunu unutmamak önemlidir. Belirli kullanım senaryonuz için optimal paralellik seviyesini belirlemek üzere kodunuzu kıyaslamak ve profillemek her zaman iyi bir uygulamadır.

Ek olarak, paralel işleme uygularken, işçiler arasındaki paylaşılan kaynakları veya bağımlılıkları göz önünde bulundurmanız gerekir. İşçilerin çakışma veya yarış koşulları olmadan bağımsız çalışabildiğinden emin olun. Bağımlılıklar veya paylaşılan kaynaklar varsa, veri bütünlüğünü korumak ve kilitlenme veya tutarsız sonuçlar gibi sorunları önlemek için uygun senkronizasyon mekanizmaları uygulamanız gerekebilir.

Ruby'nin Global Interpreter Lock'u ve Eşzamansız İşleme

Ruby'de iş parçacığı tabanlı eşzamansız işlemeyi düşünürken Ruby'nin Global Interpreter Lock (GIL) etkilerini anlamak önemlidir.

GIL, çok çekirdekli işlemcilerde bile bir seferde yalnızca bir iş parçacığının Ruby kodunu çalıştırabilmesini sağlayan Ruby yorumlayıcısındaki bir mekanizmadır. Bu, bir Ruby işlemi içinde birden çok iş parçacığı oluşturulup yönetilebilse de, herhangi bir anda yalnızca bir iş parçacığının aktif olarak Ruby kodu çalıştırabileceği anlamına gelir.

GIL, Ruby yorumlayıcısının uygulanmasını basitleştirmek ve Ruby'nin dahili veri yapıları için iş parçacığı güvenliği sağlamak üzere tasarlanmıştır. Ancak aynı zamanda Ruby kodunun gerçek paralel yürütülme potansiyelini de sınırlar.

Ruby'de `concurrent-ruby` kütüphanesi veya yerleşik `Thread` sınıfı gibi iş parçacıklarını kullandığınızda, iş parçacıkları GIL'in kısıtlamalarına tabidir. GIL, her iş parçacığının başka bir iş parçacığına geçmeden önce kısa bir süre Ruby kodu çalıştırmasına izin vererek, eşzamanlı yürütme yanılsaması yaratır.

Ancak, GIL nedeniyle, Ruby kodunun gerçek yürütülmesi sıralı kalır. Bir iş parçacığı Ruby kodu çalıştırırken, diğer iş parçacıkları esasen duraklatılır ve GIL'i edinip çalıştırmak için sıralarını bekler.

Bu, Ruby'de iş parçacığı tabanlı eşzamansız işlemenin, harici API yanıtlarını (3. taraf barındırılan büyük dil modelleri gibi) beklemek veya dosya G/Ç işlemleri gerçekleştirmek gibi G/Ç bağımlı görevler için en etkili olduğu anlamına gelir. Bir iş parçacığı bir G/Ç işlemiyle karşılaştığında, GIL'i serbest bırakabilir ve G/Ç'nin tamamlanmasını beklerken diğer iş parçacıklarının çalışmasına izin verebilir.

Öte yandan, yoğun hesaplamalar veya uzun süren AI işçi işlemeleri gibi CPU bağımlı görevler için, GIL iş parçacığı tabanlı paralelliğin potansiyel performans kazanımlarını sınırlayabilir. Bir seferde yalnızca bir iş parçacığı Ruby kodu çalıştırabildiğinden, genel yürütme süresi sıralı işlemeye kıyasla önemli ölçüde azalmayabilir.

Ruby'de CPU bağımlı görevler için gerçek paralel yürütme elde etmek için şu alternatif yaklaşımları keşfetmeniz gerekebilir:

- Her biri ayrı bir CPU çekirdeğinde çalışan birden çok Ruby işlemi ile işlem tabanlı paralellik kullanmak.
- GIL'e sahip olmayan C veya Rust gibi dillere yerel uzantılar veya arayüzler sağlayan harici kütüphaneleri veya çerçeveleri kullanmak.,
- Görevleri birden çok makine veya işlem arasında dağıtmak için dağıtık hesaplama çerçeveleri veya mesaj kuyrukları kullanmak.

Ruby'de eşzamansız işleme tasarlarken ve uygularken görevlerinizin doğasını ve GIL tarafından getirilen sınırlamaları göz önünde bulundurmak çok önemlidir. İş parçacığı tabanlı eşzamansız işleme G/Ç bağımlı görevler için faydalar sağlayabilirken, GIL'in kısıtlamaları nedeniyle CPU bağımlı görevler için önemli performans iyileştirmeleri sunmayabilir.

Gelişmiş Doğruluk için Topluluk Teknikleri

Topluluk teknikleri, sistemin genel doğruluğunu veya sağlamlığını iyileştirmek için birden çok AI işçisinin çıktılarını birleştirmeyi içerir. Tek bir AI işçisine güvenmek yerine, topluluk teknikleri daha bilinçli kararlar almak için birden çok işçinin kolektif zekasından yararlanır.

Topluluklar, iş akışınızın farklı bölümleri farklı yapay zeka modelleriyle daha iyi çalıştığında özellikle önemlidir ki bu düşündüğünüzden daha yaygın bir durumdur. GPT-4 gibi güçlü modeller, daha az yetenekli açık kaynak seçeneklerine kıyasla oldukça pahalıdır ve muhtemelen uygulamanızın her iş akışı adımı için gerekli değildir.

Yaygın bir topluluk tekniği olan çoğunluk oylaması, birden fazla yapay zeka bileşeninin aynı girdiyi bağımsız olarak işlediği ve nihai çıktının çoğunluğun konsensüsü ile

belirlendiği bir yöntemdir. Bu yaklaşım, bireysel bileşen hatalarının etkisini azaltmaya ve sistemin genel güvenilirliğini artırmaya yardımcı olabilir.

Duygu analizi için farklı modeller kullanan veya farklı bağlamlarla donatılmış üç yapay zeka bileşenine sahip olduğumuz bir örneği düşünelim. Nihai duygu tahminini belirlemek için çoğunluk oylaması kullanarak bu bileşenlerin çıktılarını birleştirebiliriz.

```
class SentimentAnalysisEnsemble
  def initialize(text)
    @text = text
  end

  def analyze
    predictions = [
      SentimentAnalysisWorker1.new(@text).analyze,
      SentimentAnalysisWorker2.new(@text).analyze,
      SentimentAnalysisWorker3.new(@text).analyze
    ]

    predictions
      .group_by { |sentiment| sentiment }
      .max_by { |_, votes| votes.size }
      .first

  end
end
```

Bu örnekte, `SentimentAnalysisEnsemble` sınıfı metin ile başlatılır ve üç farklı duygu analizi yapay zeka işçisini çağırır. `analyze` metodu her işçiden tahminleri toplar ve `group_by` ve `max_by` metotlarını kullanarak çoğunluk duygusunu belirler. Final çıktı, işçiler topluluğundan en çok oyu alan duygudur.

Topluluklar açıkça paralellik ile denemeler yapmanın değerli olabileceği bir durumdur.

Yapay Zeka İşçilerinin Dinamik Seçimi ve Çağrılması

Bazı, hatta çoğu durumda, çağrılacak belirli yapay zeka işçisi çalışma zamanı koşullarına veya kullanıcı girdilerine bağlı olabilir. Yapay zeka işçilerinin dinamik seçimi ve çağrılması, sistemde esneklik ve uyarlanabilirlik sağlar.

Kendinizi tek bir yapay zeka işçisine çok fazla işlevsellik sığdırmaya çalışırken bulabilirsiniz; ona birçok fonksiyon ve bunların nasıl çağrılacağını açıklayan büyük, karmaşık bir yönerge vermeye çalışabilirsiniz. Bu cazibeye direnin, bana güvenin. Bu bölümde tartıştığımız yaklaşımın "İşçiler Çokluğu" olarak adlandırılmasının nedenlerinden biri, daha büyük bir amaca hizmet eden, her biri kendi küçük işini yapan çok sayıda uzmanlaşmış işçiye sahip olmanın arzu edilir olduğunu hatırlatmaktır.

Örneğin, farklı yapay zeka işçilerinin farklı türdeki kullanıcı sorgularını işlemekten sorumlu olduğu bir sohbet robotu uygulamasını düşünün. Uygulama, kullanıcının girdisine bağlı olarak sorguyu işlemek için uygun yapay zeka işçisini dinamik olarak seçer.

```
class ChatbotController < ApplicationController
  def process_query
    query = params[:query]
    query_type = QueryClassifierWorker.new(query).classify

    case query_type
    when 'greeting'
      response = GreetingWorker.new(query).generate_response
    when 'product_inquiry'
      response = ProductInquiryWorker.new(query).generate_response
    when 'order_status'
      response = OrderStatusWorker.new(query).generate_response
    else
      response = DefaultResponseWorker.new(query).generate_response
    end

```

```
    render json: { response: response }
  end
end
```

Bu örnekte, `ChatbotController` kullanıcı sorgusunu `process_query` eylemi aracılığıyla alır. İlk olarak, sorgunun türünü belirlemek için bir `QueryClassifierWorker` kullanır. Sınıflandırılmış sorgu türüne bağlı olarak, denetleyici yanıtı oluşturmak için uygun YZ çalışanını dinamik olarak seçer. Bu dinamik seçim, sohbet robotunun farklı türdeki sorguları işlemesine ve bunları ilgili YZ çalışanlarına yönlendirmesine olanak tanır.

`QueryClassifierWorker`'ın işi göreceli olarak basit olduğundan ve çok fazla bağlam veya fonksiyon tanımı gerektirmediğinden, muhtemelen `mistralai/mixtral-8x7b-instruct:nitro` gibi ultra hızlı küçük bir BDM kullanarak bunu uygulayabilirsiniz. Birçok görevde GPT-4 seviyesine yakın yeteneklere sahip ve bu satırları yazdığım sırada, Groq bunu müthiş bir 444 token/saniye hızında sunabiliyor.

Geleneksel DDİ'yi BDM'lerle Birleştirme

Büyük Dil Modelleri (BDM), doğal dil işleme (DDİ) alanında devrim yaratmış ve çok çeşitli görevlerde benzersiz çok yönlülük ve performans sunmuş olsa da, her problem için her zaman en verimli veya maliyet-etkin çözüm değildir. Birçok durumda, geleneksel DDİ tekniklerini BDM'lerle birleştirmek, belirli DDİ zorluklarını çözmek için daha optimize edilmiş, hedefe yönelik ve ekonomik yaklaşımlara yol açabilir.

BDM'leri DDİ'nin İsviçre çakıları olarak düşünün—inanılmaz derecede çok yönlü ve güçlü, ancak her iş için mutlaka en iyi araç değil. Bazen, tirbuşon veya konserve açacağı gibi özel bir araç, belirli bir görev için daha etkili ve verimli olabilir. Benzer şekilde, belge kümeleme, konu tanımlama ve sınıflandırma gibi geleneksel DDİ teknikleri, DDİ

sürecinizin belirli yönleri için genellikle daha hedefe yönelik ve maliyet-etkin çözümler sunabilir.

Geleneksel DDİ tekniklerinin temel avantajlarından biri, hesaplama verimlilikleridir. Genellikle daha basit istatistiksel modellere veya kural tabanlı yaklaşımlara dayanan bu yöntemler, BDM'lere kıyasla büyük metin verilerini çok daha hızlı ve daha düşük hesaplama yüküyle işleyebilir. Bu, benzer makaleleri kümelemek veya bir metin koleksiyonu içindeki temel konuları belirlemek gibi büyük belge korpuslarını analiz etmeyi ve düzenlemeyi içeren görevler için özellikle uygundur.

Ayrıca, geleneksel DDİ teknikleri, özellikle alana özgü veri setleri üzerinde eğitildiklerinde, belirli görevler için genellikle yüksek doğruluk ve hassasiyet elde edebilir. Örneğin, Destek Vektör Makineleri (DVM) veya Naive Bayes gibi geleneksel makine öğrenimi algoritmalarını kullanan iyi ayarlanmış bir belge sınıflandırıcı, minimum hesaplama maliyetiyle belgeleri önceden tanımlanmış kategorilere doğru bir şekilde sınıflandırabilir.

Ancak, BDM'ler dil, bağlam ve akıl yürütmenin daha derin bir anlayışını gerektiren görevlerde gerçekten parlar. Tutarlı ve bağlamsal olarak ilgili metin üretme, soruları yanıtlama ve uzun pasajları özetleme yetenekleri, geleneksel DDİ yöntemleri tarafından eşlenemez. BDM'ler belirsizlik, eş referans ve deyimsel ifadeler gibi karmaşık dilbilimsel olguları etkili bir şekilde ele alabilir, bu da onları doğal dil üretimi veya anlama gerektiren görevler için paha biçilmez kılar.

Asıl güç, her ikisinin de güçlü yönlerinden yararlanan hibrit yaklaşımlar oluşturmak için geleneksel DDİ tekniklerini BDM'lerle birleştirmekte yatar. Belge ön işleme, kümeleme ve konu çıkarma gibi görevler için geleneksel DDİ yöntemlerini kullanarak, metin verilerinizi verimli bir şekilde düzenleyebilir ve yapılandırabilirsiniz. Bu yapılandırılmış bilgiler daha sonra özetler oluşturma, soruları yanıtlama veya kapsamlı raporlar oluşturma gibi daha gelişmiş görevler için BDM'lere beslenebilir.

Örneğin, büyük bir bireysel trend belgeleri korpusuna dayalı olarak belirli bir alan için bir trendler raporu oluşturmak istediğiniz bir kullanım durumunu düşünelim. Büyük

metin hacimlerini işlemek için hesaplama açısından pahalı ve zaman alıcı olabilen yalnızca BDM'lere güvenmek yerine, hibrit bir yaklaşım kullanabilirsiniz:

1. Benzer trend belgelerini gruplamak ve korpus içindeki temel temaları ve konuları belirlemek için konu modellemesi (örn. Gizli Dirichlet Tahsisi) veya kümeleme algoritmaları (örn. K-means) gibi geleneksel DDİ tekniklerini kullanın.
2. Kümelenmiş belgeleri ve tanımlanmış konuları, her küme veya konu için tutarlı ve bilgilendirici özetler oluşturmak üzere üstün dil anlama ve üretme yeteneklerinden yararlanarak bir BDM'ye besleyin.
3. Son olarak, bireysel özetleri birleştirerek, en önemli trendleri vurgulayarak ve toplanan bilgilere dayalı içgörüler ve öneriler sunarak kapsamlı bir trendler raporu oluşturmak için BDM'yi kullanın.

Geleneksel DDİ tekniklerini BDM'lerle bu şekilde birleştirerek, hesaplama kaynaklarını ve maliyetleri optimize ederken büyük miktarda metin verisini verimli bir şekilde işleyebilir, anlamlı içgörüler çıkarabilir ve yüksek kaliteli raporlar oluşturabilirsiniz.

DDİ projelerinize başlarken, her görevin özel gereksinimlerini ve kısıtlamalarını dikkatle değerlendirmek ve en iyi sonuçları elde etmek için geleneksel DDİ yöntemleri ile LLM'lerin nasıl birlikte kullanılabileceğini göz önünde bulundurmak önemlidir. Geleneksel tekniklerin verimliliğini ve hassasiyetini LLM'lerin çok yönlülüğü ve gücüyle birleştirerek, kullanıcılarınıza ve paydaşlarınıza değer katan, son derece etkili ve ekonomik DDİ çözümleri oluşturabilirsiniz.

Araç Kullanımı

Yapay zeka destekli uygulama geliştirme alanında, “araç kullanımı” veya “fonksiyon çağırma” kavramı, LLM’nizin harici araçlara, API’lere, fonksiyonlara, veritabanlarına ve diğer kaynaklara bağlanmasını sağlayan güçlü bir teknik olarak ortaya çıkmıştır. Bu yaklaşım, sadece metin çıktısı vermekten daha zengin bir davranış seti ve AI bileşenleriniz ile uygulamanızın ekosisteminin geri kalanı arasında daha dinamik etkileşimler sağlar. Bu bölümde inceleyeceğimiz gibi, araç kullanımı aynı zamanda AI modelinizin verileri yapılandırılmış şekillerde üretmesi seçeneğini de sunar.

Araç Kullanımı Nedir?

Araç kullanımı, diğer adıyla fonksiyon çağırma, geliştiricilerin LLM’nin üretim süreci sırasında etkileşimde bulunabileceği fonksiyonların bir listesini belirlemesine olanak

tanıyan bir tekniktir. Bu araçlar, basit yardımcı fonksiyonlardan karmaşık API'lere veya veritabanı sorgularına kadar uzanabilir. LLM'ye bu araçlara erişim sağlayarak, geliştiriciler modelin yeteneklerini genişletebilir ve harici bilgi veya eylem gerektiren görevleri gerçekleştirmesini sağlayabilir.

Şekil 8. Belgeleri analiz eden bir AI çalışanı için fonksiyon tanımı örneği

```
FUNCTION = {
  name: "save_analysis",
  description: "Save analysis data for document",
  parameters: {
    type: "object",
    properties: {
      title: {
        type: "string",
        maxLength: 140
      },
      summary: {
        type: "string",
        description: "comprehensive multi-paragraph summary with
                      overview and list of sections (if applicable)"
      },
      tags: {
        type: "array",
        items: {
          type: "string",
          description: "lowercase tags representing main themes
                        of the document"
        }
      }
    },
    "required": %w[title summary tags]
  }
}.freeze
```

Araç kullanımının arkasındaki temel fikir, BDM'ye kullanıcının girdisine veya eldeki göreve bağlı olarak uygun araçları dinamik olarak seçme ve yürütme yeteneği vermektir. Yalnızca modelin önceden eğitilmiş bilgisine güvenmek yerine, araç kullanımı BDM'nin daha doğru, alakalı ve eyleme geçirilebilir yanıtlar üretmek için harici kaynakları

kullanmasına olanak tanır. Araç kullanımı, GGÜ (Geri Getirme ile Güçlendirilmiş Üretim) gibi tekniklerin uygulanmasını, aksi duruma göre çok daha kolay hale getirir.

> Aksi belirtilmedikçe, bu kitap AI modelinizin herhangi bir yerleşik sunucu tarafı araca erişimi olmadığını varsayar. AI'nızın kullanımına sunmak istediğiniz her aracın, AI'nız size bu aracı yanıtında kullanmak istediğini söylediğinde yürütülmesini sağlayacak önlemlerle birlikte, her API isteğinde açıkça sizin tarafınızdan belirtilmesi gerekir.

Araç Kullanımının Potansiyeli

Araç kullanımı, AI destekli uygulamalar için geniş bir olasılıklar yelpazesi açar. İşte araç kullanımıyla neler başarılabileceğine dair birkaç örnek:

1. **Sohbet Botları ve Sanal Asistanlar**: Bir BDM'yi harici araçlara bağlayarak, sohbet botları ve sanal asistanlar veritabanlarından bilgi alma, API çağrıları yapma veya diğer sistemlerle etkileşim kurma gibi daha karmaşık görevleri gerçekleştirebilir. Örneğin, bir sohbet botu kullanıcının isteğine bağlı olarak bir anlaşmanın durumunu değiştirmek için bir MİY aracını kullanabilir.
2. **Veri Analizi ve İçgörüler**: BDM'ler gelişmiş veri işleme görevlerini gerçekleştirmek için veri analizi araçlarına veya kütüphanelerine bağlanabilir. Bu, uygulamaların kullanıcı sorgularına dayalı olarak içgörüler üretmesini, karşılaştırmalı analizler yapmasını veya veriye dayalı öneriler sunmasını sağlar.
3. **Arama ve Bilgi Getirme**: Araç kullanımı, BDM'lerin arama motorları, vektör veritabanları veya diğer bilgi getirme sistemleriyle etkileşim kurmasına olanak tanır. BDM, kullanıcı sorgularını arama sorgularına dönüştürerek birden fazla kaynaktan ilgili bilgileri getirebilir ve kullanıcı sorularına kapsamlı yanıtlar sağlayabilir.

4. **Harici Hizmetlerle Entegrasyon**: Araç kullanımı, AI destekli uygulamalar ile harici hizmetler veya API'ler arasında sorunsuz entegrasyon sağlar. Örneğin, bir BDM gerçek zamanlı hava durumu güncellemeleri sağlamak için bir hava durumu API'si ile veya çok dilli yanıtlar üretmek için bir çeviri API'si ile etkileşime girebilir.

Araç Kullanım İş Akışı

Araç kullanım iş akışı genellikle dört temel adımı içerir:

1. İstek bağlamına fonksiyon tanımlarını dahil etme
2. Dinamik (veya açık) araç seçimi
3. Fonksiyon(lar)ın yürütülmesi
4. Orijinal isteğin isteğe bağlı devamı

Bu adımların her birini detaylı olarak inceleyelim.

İstek bağlamına fonksiyon tanımlarını dahil etme

AI, tamamlama isteğinizin bir parçası olarak verdiğiniz bir liste sayesinde (genellikle JSON şeması varyantı kullanılarak tanımlanan fonksiyonlar) hangi araçlara sahip olduğunu bilir.

Araç tanımlama sözdizimi modele özgüdür.

Claude 3'te bir `get_weather` fonksiyonunu şu şekilde tanımlarsınız:

```
{
    "name": "get_weather",
    "description": "Get the current weather in a given location",
    "input_schema": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA"
            },
            "unit": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "description": "The unit of temperature"
        }
    },
    "required": ["location"]
    }
}
```

Ve aynı fonksiyonu GPT-4 için tanımlamak için, `tools` parametresinin değeri olarak şu şekilde iletirsiniz:

```
{
    "name": "get_current_weather",
    "description": "Get the current weather in a given location",
    "parameters": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA",
            },
            "unit": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "description": "The unit of temperature"
            },
        },
        "required": ["location"],
```

```
    },
}
```

> Neredeyse aynı, ama görünürde hiçbir sebep yokken farklı! Ne kadar sinir bozucu.

Fonksiyon tanımlamaları isim, açıklama ve giriş parametrelerini belirtir. Giriş parametreleri, kabul edilebilir değerleri sınırlamak için enum gibi öznitelikler kullanılarak ve bir parametrenin gerekli olup olmadığı belirtilerek daha ayrıntılı tanımlanabilir.

Gerçek fonksiyon tanımlamalarına ek olarak, sistem yönergesine fonksiyonun sistemde neden ve nasıl kullanılacağına dair talimatları veya bağlamı da ekleyebilirsiniz.

Örneğin, Olympia'daki Web Arama aracım, YZ'ye bahsedilen araçların kullanımında olduğunu hatırlatan şu sistem yönergesini içerir:

```
The `google_search` and `realtime_search` functions let you do research
on behalf of the user. In contrast to Google, realtime search is powered
by Perplexity and provides real-time information to curated current events
databases and news sources. Make sure to include URLs in your response so
user can do followup research.
```

Detaylı açıklamalar sağlamak, araç performansında en önemli faktör olarak kabul edilir. Açıklamalarınız, araç hakkındaki her detayı açıklamalıdır, bunlar dahil:

- Aracın ne yaptığı
- Ne zaman kullanılması gerektiği (ve ne zaman kullanılmaması gerektiği)
- Her parametrenin ne anlama geldiği ve aracın davranışını nasıl etkilediği
- Aracın uygulamasına ilişkin önemli uyarılar veya kısıtlamalar

Araçlarınız hakkında yapay zekaya ne kadar çok bağlam sağlarsanız, yapay zeka bu araçları ne zaman ve nasıl kullanacağına o kadar iyi karar verecektir. Örneğin,

Anthropic, Claude 3 serisi için araç başına en az 3-4 cümlelik açıklama önermektedir, eğer araç karmaşıksa daha fazlası gerekebilir.

Sezgisel olmayabilir, ancak açıklamalar örneklerden daha önemli kabul edilir. Bir aracın nasıl kullanılacağına dair örnekleri açıklamasına veya beraberindeki yönergeye ekleyebilirseniz de, bu, aracın amacının ve parametrelerinin net ve kapsamlı bir açıklamasına sahip olmaktan daha az önemlidir. Örnekleri ancak açıklamayı tam olarak geliştirdikten sonra ekleyin.

İşte Stripe benzeri bir API fonksiyon spesifikasyonu örneği:

```
{
  "name": "createPayment",
  "description": "Create a new payment request",
  "parameters": {
    "type": "object",
    "properties": {
      "transaction_amount": {
        "type": "number",
        "description": "The amount to be paid"
      },
      "description": {
        "type": "string",
        "description": "A brief description of the payment"
      },
      "payment_method_id": {
        "type": "string",
        "description": "The payment method to be used"
      },
      "payer": {
        "type": "object",
        "description": "Information about the payer, including their name,
                        email, and identification number",
        "properties": {
          "name": {
            "type": "string",
            "description": "The payer's name"
        },
        "email": {
          "type": "string",
```

```
            "description": "The payer's email address"
          },
          "identification": {
            "type": "object",
            "description": "The payer's identification number",
            "properties": {
              "type": {
                "type": "string",
                "description": "Identification document (e.g. CPF, CNPJ)"
              },
              "number": {
                "type": "string",
                "description": "The identification number"
              }
            },
            "required": [ "type", "number" ]
          }
        },
        "required": [ "name", "email", "identification" ]
      }
    }
}
```

Pratikte, bazı modeller iç içe geçmiş fonksiyon tanımlamalarıyla ve diziler, sözlükler gibi karmaşık çıktı veri türleriyle başa çıkmakta zorlanır. Ancak teoride, herhangi bir derinlikte JSON Şema tanımlamaları sağlayabilmelisiniz!

Dinamik Araç Seçimi

Araç tanımlamaları içeren bir sohbet tamamlaması yürüttüğünüzde, DDD dinamik olarak kullanılacak en uygun araç(ları) seçer ve her araç için gerekli girdi parametrelerini oluşturur.

Pratikte, yapay zekanın *tam olarak* doğru fonksiyonu çağırma ve girdi tanımlamalarınıza *tam olarak* uyma kapasitesi değişkenlik gösterir. Sıcaklık

hiperparametresini 0.0'a kadar düşürmek çok yardımcı olur, ancak deneyimlerime göre yine de ara sıra hatalarla karşılaşırsınız. Bu hatalar arasında hayal edilmiş fonksiyon isimleri, yanlış isimlendirilmiş veya tamamen eksik girdi parametreleri bulunur. Parametreler JSON olarak iletilir, bu da bazen kesik, yanlış tırnak işaretli veya başka şekillerde bozuk JSON'dan kaynaklanan hatalar görebileceğiniz anlamına gelir.

Kendi Kendini Onaran Veri desenleri, sözdizimi hatalarından dolayı bozulan fonksiyon çağrılarını otomatik olarak düzeltmeye yardımcı olabilir.

Zorlanmış (Açık) Araç Seçimi

Bazı modeller, istekte bir parametre olarak belirli bir fonksiyonun çağrılmasını zorlama seçeneği sunar. Aksi takdirde, fonksiyonun çağrılıp çağrılmayacağı tamamen yapay zekanın takdirine bağlıdır.

Bir fonksiyon çağrısını zorlama yeteneği, yapay zekanın dinamik seçim sürecinden bağımsız olarak belirli bir aracın veya fonksiyonun yürütülmesini sağlamak istediğiniz bazı senaryolarda çok önemlidir. Bu yeteneğin önemli olmasının birkaç nedeni vardır:

1. **Açık Kontrol**: Yapay zekayı bir *Ayrık Bileşen* olarak veya belirli bir zamanda belirli bir fonksiyonun yürütülmesini gerektiren önceden tanımlanmış bir iş akışında kullanıyor olabilirsiniz. Çağrıyı zorlayarak, yapay zekadan nazikçe yapmasını istemek yerine istenen fonksiyonun çağrılmasını garanti edebilirsiniz.
2. **Hata Ayıklama ve Test**: Yapay zeka destekli uygulamaları geliştirirken ve test ederken, fonksiyon çağrılarını zorlama yeteneği hata ayıklama amaçları için çok değerlidir. Belirli fonksiyonları açıkça tetikleyerek, uygulamanızın bileşenlerini izole edebilir ve test edebilirsiniz. Bu, fonksiyon uygulamalarının doğruluğunu doğrulamanıza, girdi parametrelerini kontrol etmenize ve beklenen sonuçların döndürüldüğünden emin olmanıza olanak tanır.

3. **Uç Durumları Ele Alma**: Yapay zekanın dinamik seçim sürecinin, dış süreçlere dayalı olarak çağırması gerektiğini bildiğiniz bir fonksiyonu seçmeyebileceği uç durumlar veya istisnai senaryolar olabilir. Bu gibi durumlarda, bir fonksiyon çağrısını zorlama yeteneği bu durumları açıkça ele almanıza olanak tanır. Yapay zekanın takdirini ne zaman geçersiz kılacağınızı belirlemek için uygulama mantığınızda kurallar veya koşullar tanımlayın.
4. **Tutarlılık ve Tekrarlanabilirlik**: Belirli bir sırayla yürütülmesi gereken belirli bir fonksiyon diziniz varsa, çağrıları zorlamak her seferinde aynı sıranın izlenmesini garanti eder. Bu, finansal sistemler veya bilimsel simülasyonlar gibi tutarlılığın ve öngörülebilir davranışın kritik olduğu uygulamalarda özellikle önemlidir.
5. **Performans Optimizasyonu**: Bazı durumlarda, bir fonksiyon çağrısını zorlamak performans optimizasyonlarına yol açabilir. Belirli bir görev için belirli bir fonksiyonun gerekli olduğunu ve yapay zekanın dinamik seçim sürecinin gereksiz ek yük getirebileceğini biliyorsanız, seçim sürecini atlayabilir ve gerekli fonksiyonu doğrudan çağırabilirsiniz. Bu, uygulamanızın genel verimliliğini artırabilir ve gecikmeyi azaltabilir.

Özetle, yapay zeka destekli uygulamalarda fonksiyon çağrılarını zorlama yeteneği açık kontrol sağlar, hata ayıklama ve teste yardımcı olur, uç durumları ele alır, tutarlılık ve tekrarlanabilirlik sağlar. Bu, cephanenizdeki güçlü bir araçtır, ancak bu önemli özelliğin bir yönünü daha tartışmamız gerekiyor.

Birçok karar verme kullanım durumunda, modelin her zaman bir fonksiyon çağrısı yapmasını isteyebilir ve modelin asla sadece kendi iç bilgisiyle yanıt vermesini istemeyebiliriz. Örneğin, farklı görevlerde (çok dilli girdi, matematik vb.) uzmanlaşmış birden çok model arasında yönlendirme yapıyorsanız, fonksiyon çağırma modelini istekleri yardımcı modellerden birine yönlendirmek için kullanabilir ve asla bağımsız olarak yanıt vermeyebilirsiniz.

Araç Seçimi Parametresi

GPT-4 ve fonksiyon çağırmayı destekleyen diğer dil modelleri, bir tamamlamanın parçası olarak araç kullanımının gerekli olup olmadığını kontrol etmek için size bir `tool_choice` parametresi sunar. Bu parametrenin üç olası değeri vardır:

- `auto` yapay zekaya bir araç kullanma veya basitçe yanıt verme konusunda tam takdir yetkisi verir
- `required` yapay zekaya bir araç çağırması *gerektiğini* söyler, ancak aracın seçimini yapay zekaya bırakır
- Üçüncü seçenek, zorlamak istediğiniz `name_of_function` parametresini ayarlamaktır. Bununla ilgili daha fazla bilgi bir sonraki bölümde.

`tool_choice` değerini `required` olarak ayarlarsanız, modelin verilen fonksiyonlar arasından, hiçbiri istenen işe tam olarak uymasa bile, en uygun olanını seçmek zorunda kalacağını unutmayın. Yayın tarihi itibarıyla, boş bir `tool_calls` yanıtı döndüren veya uygun bir fonksiyon bulamadığını başka bir şekilde bildiren bir model bilmiyorum.

Yapılandırılmış Çıktı İçin Fonksiyon Kullanımını Zorunlu Kılma

Fonksiyon çağrısını zorunlu kılma özelliği, düz metin yanıtından kendiniz çıkarmak yerine, sohbet tamamlamadan yapılandırılmış veri almanın bir yolunu sunar.

Yapılandırılmış çıktı almak için fonksiyonları zorlamanın neden bu kadar önemli olduğunu merak ediyor musunuz? Basitçe söylemek gerekirse, BDM çıktısından yapılandırılmış veri çıkarmak oldukça zahmetli bir iştir. Verileri XML formatında isteyerek işinizi biraz kolaylaştırabilirsiniz, ancak bu sefer de XML ayrıştırması

> yapmanız gerekir. Peki ya YZ size "Üzgünüm, şu şu sebeplerden dolayı istediğiniz veriyi oluşturamıyorum..." diye yanıt verdiğinde ve XML eksik kaldığında ne yaparsınız?

Araçları bu şekilde kullanırken:

- Muhtemelen isteğinizde tek bir araç tanımlamalısınız
- `tool_choice` parametresini kullanarak fonksiyonun kullanımını zorlamayı unutmayın
- Model girdiyi araca ileteceği için, aracın adı ve açıklaması sizin bakış açınızdan değil, modelin bakış açısından olmalıdır.

Bu son nokta açıklık getirmek için bir örneği hak ediyor. Diyelim ki YZ'den kullanıcı metninde duygu analizi yapmasını istiyorsunuz. Fonksiyonun adı `analyze_sentiment` değil, `save_sentiment_analysis` gibi bir şey olmalıdır. Duygu analizini yapan YZ'dir, *araç değil*. Aracın yaptığı tek şey (YZ'nin bakış açısından) analiz sonuçlarını kaydetmektir.

İşte Claude 3 kullanarak bir görüntünün özetini iyi yapılandırılmış JSON formatında kaydetmenin bir örneği, bu sefer komut satırından `curl` kullanarak:

```
curl https://api.anthropic.com/v1/messages \
     --header "content-type: application/json" \
     --header "x-api-key: $ANTHROPIC_API_KEY" \
     --header "anthropic-version: 2023-06-01" \
     --header "anthropic-beta: tools-2024-04-04" \
     --data \
'{
    "model": "claude-3-sonnet-20240229",
    "max_tokens": 1024,
    "tools": [{
        "name": "record_summary",
        "description": "Record summary of image into well-structured JSON.",
        "input_schema": {
            "type": "object",
            "properties": {
                "key_colors": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "r": {
                                "type": "number",
                                "description": "red value [0.0, 1.0]"
                            },
                            "g": {
                                "type": "number",
                                "description": "green value [0.0, 1.0]"
                            },
                            "b": {
                                "type": "number",
                                "description": "blue value [0.0, 1.0]"
                            },
                            "name": {
                                "type": "string",
                                "description": "Human-readable color name
                                                in snake_case, e.g.
                                                \"olive_green\"or
                                                \"turquoise\""
                            }
                        },
                        "required": [ "r", "g", "b", "name" ]
                    },
```

```
                    "description": "Key colors in the image. Four or less."
                },
                "description": {
                    "type": "string",
                    "description": "Image description. 1-2 sentences max."
                },
                "estimated_year": {
                    "type": "integer",
                    "description": "Estimated year that the image was taken,
                                    if is it a photo. Only set this if the
                                    image appears to be non-fictional.
                                    Rough estimates are okay!"
                }
            },
            "required": [ "key_colors", "description" ]
        }
    }],
    "messages": [
        {
            "role": "user",
            "content": [
                {
                    "type": "image",
                    "source": {
                        "type": "base64",
                        "media_type": "'$IMAGE_MEDIA_TYPE'",
                        "data": "'$IMAGE_BASE64'"
                    }
                },
                {
                    "type": "text",
                    "text": "Use `record_summary` to describe this image."
                }
            ]
        }
    ]
}'
```

Verilen örnekte, bir görüntünün yapılandırılmış JSON özetini oluşturmak için Anthropic'ten Claude 3 modelini kullanıyoruz. İşte nasıl çalışıyor:

1. İstek yükünün `tools` dizisinde `record_summary` adında tek bir araç tanımlıyoruz. Bu araç, görüntünün özetini iyi yapılandırılmış JSON formatında kaydetmekten sorumludur.
2. `record_summary` aracı, JSON çıktısının beklenen yapısını belirten bir `input_schema`'ya sahiptir. Üç özellik tanımlar:

    - `key_colors`: Görüntüdeki ana renkleri temsil eden nesneler dizisi. Her renk nesnesi, kırmızı, yeşil ve mavi değerleri için özellikler (0.0 ile 1.0 arasında) ve snake_case formatında insan tarafından okunabilir bir renk adına sahiptir.
    - `description`: Görüntünün 1-2 cümlelik kısa açıklaması için bir string özelliği.
    - `estimated_year`: Kurgu dışı bir fotoğraf gibi görünüyorsa, görüntünün çekildiği tahmini yıl için isteğe bağlı bir tamsayı özelliği.

3. `messages` dizisinde, görüntü verisini base64 kodlu bir string olarak medya türüyle birlikte sağlıyoruz. Bu, modelin görüntüyü girdi olarak işlemesine olanak tanır.
4. Ayrıca Claude'a görüntüyü tanımlamak için `record_summary` aracını kullanması için yönlendirme yapıyoruz.
5. İstek Claude 3 modeline gönderildiğinde, model görüntüyü analiz eder ve belirtilen `input_schema`'ya göre bir JSON özeti oluşturur. Model ana renkleri çıkarır, kısa bir açıklama sağlar ve varsa görüntünün çekildiği yılı tahmin eder.
6. Oluşturulan JSON özeti, `record_summary` aracına parametre olarak iletilir ve görüntünün temel özelliklerinin yapılandırılmış bir temsilini sağlar.

`record_summary` aracını iyi tanımlanmış bir `input_schema` ile kullanarak, düz metin çıkarımına güvenmeden bir görüntünün yapılandırılmış JSON özetini elde edebiliriz. Bu yaklaşım, çıktının tutarlı bir formatı takip etmesini sağlar ve uygulamanın alt sistem bileşenleri tarafından kolayca ayrıştırılıp işlenebilir.

Bir fonksiyon çağrısını zorlamak ve beklenen çıktı yapısını belirlemek, AI tabanlı uygulamalardaki araç kullanımının güçlü bir özelliğidir. Bu, geliştiricilerin oluşturulan

çıktı üzerinde daha fazla kontrol sahibi olmasını sağlar ve AI tarafından oluşturulan verilerin uygulama iş akışına entegrasyonunu basitleştirir.

Fonksiyon(lar)ın Yürütülmesi

Fonksiyonları tanımladınız ve AI'nızı yönlendirdiniz, AI de fonksiyonlarınızdan birini çağırması gerektiğine karar verdi. Şimdi sıra, `raix-rails` gibi bir Ruby gem kullanıyorsanız, uygulama kodunuzun veya kütüphanenizin fonksiyon çağrısını ve parametrelerini *uygulama kodunuzdaki* karşılık gelen uygulamaya göndermesinde.

Uygulama kodunuz, fonksiyon yürütme sonuçlarıyla ne yapılacağına karar verir. Belki yapılacak şey bir lambda içindeki tek satırlık bir kod olabilir, veya belki harici bir API'yi çağırmayı içerebilir. Belki başka bir AI bileşenini çağırmayı içerebilir, veya belki sisteminizin geri kalanında yüzlerce hatta binlerce satır kodu içerebilir. Bu tamamen size bağlı.

Bazen fonksiyon çağrısı işlemin sonu olur, ancak sonuçlar AI tarafından devam ettirilecek bir düşünce zincirindeki bilgileri temsil ediyorsa, uygulama kodunuzun yürütme sonuçlarını sohbet dökümüne eklemesi ve AI'nın işlemeye devam etmesine izin vermesi gerekir.

Örneğin, müşteri hizmetleri için Akıllı İş Akışı Orkestrasyon'un bir parçası olarak Olympia'nın `AccountManager`'ının müşterilerimizle iletişim kurmak için kullandığı bir Raix fonksiyon tanımı.

```ruby
class AccountManager
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  # lots of other functions...

  function :notify_account_owner,
    "Don't share UUID. Mention dollars if subscription changed",
    message: { type: "string" } do |arguments|
      account.owner.freeform_notify(
        subject: "Account Change Notification",
        message: arguments[:message]
      )
      "Notified account owner"
    end
```

Burada ne olduğu ilk bakışta açık olmayabilir, bu yüzden parça parça açıklayacağım.

1. `AccountManager` sınıfı hesap yönetimiyle ilgili birçok fonksiyon tanımlar. Planınızı değiştirebilir, ekip üyelerini ekleyip çıkarabilir ve başka işlemler yapabilir.
2. Üst düzey talimatları, `AccountManager`'a hesap değişikliği isteğinin sonuçlarını `notify_account_owner` fonksiyonunu kullanarak hesap sahibine bildirmesi gerektiğini söyler.
3. Fonksiyonun özlü tanımı şunları içerir:

- adı
- açıklaması
- parametreleri `message: { type: "string" }`
- fonksiyon çağrıldığında yürütülecek bir blok

Fonksiyon bloğunun sonuçlarıyla döküm güncellendikten sonra, `chat_completion` metodu tekrar çağrılır. Bu metod, güncellenmiş konuşma dökümünü daha fazla işlem için AI modeline geri göndermekten sorumludur. Bu süreci *konuşma döngüsü* olarak adlandırıyoruz.

AI modeli güncellenmiş dökümle yeni bir sohbet tamamlama isteği aldığında, önceden yürütülen fonksiyonun sonuçlarına erişimi olur. Bu sonuçları analiz edebilir, karar verme sürecine dahil edebilir ve konuşmanın birikimli bağlamına dayalı olarak bir sonraki yanıtı veya eylemi oluşturabilir. Güncellenmiş bağlama göre ek fonksiyonlar yürütmeyi seçebilir veya başka fonksiyon çağrılarının gerekli olmadığına karar verirse orijinal isteme son bir yanıt oluşturabilir.

Orijinal İsteğin İsteğe Bağlı Devamı

Araç sonuçlarını BDM'ye geri gönderdiğinizde ve orijinal isteğin işlenmesine devam ettiğinizde, AI bu sonuçları ek fonksiyonlar çağırmak veya düz metin şeklinde son bir yanıt oluşturmak için kullanır.

Cohere'in Command-R gibi bazı modeller, yanıtlarında kullandıkları belirli araçları belirtebilir, bu da ek şeffaflık ve izlenebilirlik sağlar.

Kullanılan modele bağlı olarak, fonksiyon çağrısının sonuçları kendi özel rolüne sahip döküm mesajlarında yaşayacak veya başka bir sözdiziminde yansıtılacaktır. Ancak önemli olan bu verilerin, AI'nın bir sonraki adımda ne yapacağına karar verirken değerlendirebilmesi için dökümde bulunmasıdır.

Yaygın (ve potansiyel olarak pahalı) bir hata durumu, sohbete devam etmeden önce fonksiyon sonuçlarını döküme eklemeyi unutmaktır. Sonuç olarak, AI neredeyse fonksiyonu ilk kez çağırmadan önceki gibi aynı şekilde uyarılacaktır. Başka bir deyişle, AI'nın bakış açısından, henüz fonksiyonu çağırmamıştır. Bu yüzden tekrar çağırır. Ve tekrar. Ve siz durdurana kadar tekrar eder. Umarım bağlamınız çok büyük değildir ve modeliniz çok pahalı değildir!

Araç Kullanımı İçin En İyi Uygulamalar

Araç kullanımından en iyi şekilde yararlanmak için aşağıdaki en iyi uygulamaları göz önünde bulundurun.

Açıklayıcı Tanımlar

Her araç ve giriş parametreleri için açık ve açıklayıcı isimler ve tanımlar sağlayın. Bu, BDM'nin her aracın amacını ve yeteneklerini daha iyi anlamasına yardımcı olur.

> Deneyimlerimden söyleyebilirim ki "isimlendirmenin zor olduğu" yönündeki yaygın görüş burada da geçerli; sadece fonksiyonların isimlerini veya açıklamaların ifade edilişini değiştirerek BDM'lerden dramatik şekilde farklı sonuçlar aldığımı gördüm. Bazen açıklamaları kaldırmak performansı *iyileştirir*.

Araç Sonuçlarının İşlenmesi

Araç sonuçlarını BDM'ye geri gönderirken, bunların iyi yapılandırılmış ve kapsamlı olmasını sağlayın. Her aracın çıktısını temsil etmek için anlamlı anahtarlar ve değerler kullanın. JSON'dan düz metne kadar farklı formatlarla denemeler yapın ve hangisinin en iyi çalıştığını görün.

Sonuç Yorumlayıcı, sonuçları analiz etmek ve insan dostu açıklamalar, özetler veya önemli çıkarımlar sağlamak için AI kullanarak bu zorluğu ele alır.

Hata Yönetimi

BDM'nin araç çağrıları için geçersiz veya desteklenmeyen giriş parametreleri oluşturabileceği durumları ele almak için sağlam hata yönetimi mekanizmaları

uygulayın. Araç yürütmesi sırasında oluşabilecek herhangi bir hatayı düzgün bir şekilde ele alın ve bu hatalardan kurtulun.

AI'nın çok güzel bir özelliği, hata mesajlarını anlıyor olması! Bu da demek oluyor ki, hızlı ve pratik bir yaklaşımla çalışıyorsanız, bir aracın uygulamasında oluşan herhangi bir istisnayı yakalayıp ne olduğunu anlaması için AI'ya geri gönderebilirsiniz!

Örneğin, işte Olympia'daki Google aramasının uygulamasının sadeleştirilmiş bir versiyonu:

```
1   def google_search(conversation, params)
2     conversation.update_cstatus("Searching Google...")
3     query = params[:query]
4     search = GoogleSearch.new(query).get_hash
5
6     conversation.update_cstatus("Summarizing results...")
7     SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
8   rescue StandardError => e
9     Honeybadger.notify(e)
10    { error: e.message }.inspect
11  end
```

Olympia'daki Google aramaları iki aşamalı bir süreçtir. Önce aramayı yaparsınız, sonra sonuçları özetlersiniz. Herhangi bir başarısızlık durumunda, ne olursa olsun, istisna mesajı paketlenir ve yapay zekaya geri gönderilir. Bu teknik, neredeyse tüm *Akıllı Hata İşleme* desenlerinin temelidir.

Örneğin, `GoogleSearch` API çağrısının 503 Hizmet Kullanılamıyor istisnası nedeniyle başarısız olduğunu varsayalım. Bu hata en üst seviyedeki kurtarma bloğuna kadar yükselir ve hatanın açıklaması fonksiyon çağrısının sonucu olarak yapay zekaya geri gönderilir. Kullanıcıya sadece boş bir ekran veya teknik bir hata vermek yerine, yapay zeka "Üzgünüm, şu anda Google Arama özelliklerime erişemiyorum. İsterseniz daha sonra tekrar deneyebilirim." gibi bir şey söyler.

Bu sadece akıllıca bir numara gibi görünebilir, ancak farklı türde bir hatayı düşünün, yapay zekanın harici bir API'yi çağırdığı ve API'ye geçirilecek parametreler üzerinde

doğrudan kontrole sahip olduğu bir durumu. Belki bu parametreleri oluştururken bir hata mı yaptı? Harici API'den gelen hata mesajı yeterince detaylı olduğu sürece, hata mesajını çağıran yapay zekaya geri göndermek, onun bu parametreleri yeniden değerlendirmesine ve tekrar denemesine olanak sağlar. Otomatik olarak. Hata ne olursa olsun.

Şimdi bu tür sağlam hata işlemenin *normal* kodda çoğaltılması için ne gerektiğini düşünün. Bu neredeyse imkansızdır.

Yinelemeli İyileştirme

Eğer LLM uygun araçları önermiyor veya optimal olmayan yanıtlar üretiyorsa, araç tanımları, açıklamaları ve giriş parametreleri üzerinde yineleme yapın. Gözlemlenen davranışa ve istenen sonuçlara dayalı olarak araç kurulumunu sürekli olarak iyileştirin ve geliştirin.

1. Basit araç tanımlarıyla başlayın: Açık ve özlü isimler, açıklamalar ve giriş parametreleriyle araçları tanımlamaya başlayın. Başlangıçta araç kurulumunu karmaşık hale getirmekten kaçının ve temel işlevselliğe odaklanın. Örneğin, duygu analizi sonuçlarını kaydetmek istiyorsanız, şöyle temel bir tanımla başlayın:

```
{
  "name": "save_sentiment_score",
  "description": "Analyze user-provided text and generate sentiment score",
  "parameters": {
    "type": "object",
    "properties": {
      "score": {
        "type": "float",
        "description": "sentiment score from -1 (negative) to 1 (positive)"
      }
    },
    "required": ["score"]
  }
}
```

2. Test et ve gözlemle: İlk araç tanımlamalarını yaptıktan sonra, bunları farklı girdilerle test et ve BDM'nin araçla nasıl etkileşime girdiğini gözlemle. Üretilen yanıtların kalitesine ve uygunluğuna dikkat et. Eğer BDM optimal altı yanıtlar üretiyorsa, araç tanımlarını iyileştirme zamanı gelmiş demektir.
3. Tanımları iyileştir: Eğer BDM bir aracın amacını yanlış anlıyorsa, aracın tanımını iyileştirmeyi dene. BDM'nin aracı etkili bir şekilde kullanmasına yardımcı olmak için daha fazla bağlam, örnek veya açıklama sağla. Örneğin, duygu analizi aracının tanımını, analiz edilen metnin *duygusal tonuna* daha özel olarak değinecek şekilde güncelleyebilirsin:

```
{
  "name": "save_sentiment_score",
  "description": "Determine the overall emotional tone of a piece of text,
   such as customer reviews, social media posts, or feedback comments.",
  ...
}
```

4. Giriş parametrelerini ayarlayın: Eğer LLM bir araç için geçersiz veya alakasız giriş parametreleri üretiyorsa, parametre tanımlamalarını ayarlamayı düşünün. Beklenen giriş formatını netleştirmek için daha spesifik kısıtlamalar, doğrulama kuralları veya örnekler ekleyin.

5. Geri bildirimlere göre yineleyin: Araçlarınızın performansını sürekli olarak izleyin ve kullanıcılardan veya paydaşlardan geri bildirim toplayın. Bu geri bildirimleri kullanarak iyileştirmeye açık alanları belirleyin ve araç tanımlamalarında tekrarlayan iyileştirmeler yapın. Örneğin, kullanıcılar analizin alaycılığı iyi işlemediğini bildirirse, açıklamaya bir not ekleyebilirsiniz:

```
{
  "name": "save_sentiment_score",
  "description": "Analyze the sentiment of a given text and return a sentiment
   score between -1 (negative) and 1 (positive). Note: Sarcasm should be
   considered negative.",
  ...
}
```

Araç tanımlarınızı gözlemlenen davranış ve geri bildirimlere dayalı olarak yinelemeli bir şekilde geliştirerek, yapay zeka destekli uygulamanızın performansını ve etkinliğini kademeli olarak iyileştirebilirsiniz. Araç tanımlarını net, özlü ve belirli göreve odaklı tutmayı unutmayın. İstediğiniz sonuçlarla uyumlu olduklarından emin olmak için araç etkileşimlerini düzenli olarak test edin ve doğrulayın.

Araçları Birleştirme ve Zincirleme

Şimdiye kadar sadece değinilen araç kullanımının en güçlü yönlerinden biri, karmaşık görevleri gerçekleştirmek için birden fazla aracı birleştirme ve zincirleme yapabilme yeteneğidir. Araç tanımlarınızı ve bunların girdi/çıktı formatlarını dikkatli bir şekilde tasarlayarak, çeşitli şekillerde birleştirilebilen yeniden kullanılabilir yapı taşları oluşturabilirsiniz.

Yapay zeka destekli uygulamanız için bir veri analizi hattı oluşturduğunuz bir örneği düşünelim. Aşağıdaki araçlara sahip olabilirsiniz:

1. **`DataRetrieval`**: Belirtilen kriterlere göre bir veritabanından veya API'den veri çeken bir araç.

2. **`DataProcessing`**: Alınan veriler üzerinde hesaplamalar, dönüşümler veya toplamalar gerçekleştiren bir araç.
3. **`DataVisualization`**: İşlenmiş verileri grafikler veya şemalar gibi kullanıcı dostu bir formatta sunan bir araç.

Bu araçları birbirine zincirleyerek, ilgili verileri alan, işleyen ve sonuçları anlamlı bir şekilde sunan güçlü bir iş akışı oluşturabilirsiniz. Araç kullanım iş akışı şöyle görünebilir:

1. BDM, belirli bir ürün kategorisi için satış verileri hakkında içgörüler isteyen bir kullanıcı sorgusu alır.
2. BDM, `DataRetrieval` aracını seçer ve veritabanından ilgili satış verilerini almak için uygun girdi parametrelerini oluşturur.
3. Alınan veriler, toplam gelir, ortalama satış fiyatı ve büyüme oranı gibi metrikleri hesaplayan `DataProcessing` aracına "aktarılır".
4. İşlenen veriler daha sonra `DataVisualization` aracı tarafından işlenir ve içgörüleri temsil eden görsel açıdan çekici bir grafik veya şema oluşturulur, grafiğin URL'si BDM'ye geri iletilir.
5. Son olarak, BDM, görselleştirilmiş verileri ve temel bulguların bir özetini içeren, markdown kullanarak biçimlendirilmiş bir yanıt oluşturur.

Bu araçları bir araya getirerek, uygulamanıza kolayca entegre edilebilen kesintisiz bir veri analizi iş akışı oluşturabilirsiniz. Bu yaklaşımın güzelliği, her aracın bağımsız olarak geliştirilebilmesi ve test edilebilmesi, ardından çeşitli sorunları çözmek için farklı şekillerde birleştirilebilmesidir.

Araçların sorunsuz birleştirilmesini ve zincirlenmesini sağlamak için, her araç için net girdi ve çıktı formatları tanımlamak önemlidir.

Örneğin, `DataRetrieval` aracı, veritabanı bağlantı detayları, tablo adı ve sorgu koşulları gibi parametreleri kabul edebilir ve sonuç kümesini yapılandırılmış bir JSON nesnesi olarak döndürebilir. `DataProcessing` aracı daha sonra bu JSON nesnesini

girdi olarak bekleyebilir ve dönüştürülmüş bir JSON nesnesi çıktısı üretebilir. Araçlar arasındaki veri akışını standartlaştırarak uyumluluk ve yeniden kullanılabilirliği sağlayabilirsiniz.

Araç ekosisteminizi tasarlarken, farklı araçların uygulamanızdaki yaygın kullanım senaryolarını ele almak için nasıl birleştirilebileceğini düşünün. BDM'nin etkili bir şekilde seçip kullanmasını kolaylaştırmak için yaygın iş akışlarını veya iş mantığını kapsayan üst düzey araçlar oluşturmayı düşünün.

Unutmayın, araç kullanımının gücü, sağladığı esneklik ve modülerlikte yatar. Karmaşık görevleri daha küçük, yeniden kullanılabilir araçlara bölerek, çok çeşitli zorluklarla başa çıkabilen sağlam ve uyarlanabilir bir yapay zeka destekli uygulama oluşturabilirsiniz.

Gelecekteki Yönelimler

Yapay zeka destekli uygulama geliştirme alanı geliştikçe, araç kullanım yeteneklerinde daha fazla ilerleme bekleyebiliriz. Bazı potansiyel gelecek yönelimleri şunları içerir:

1. **Çok Aşamalı Araç Kullanımı**: BDM'ler, tatmin edici bir yanıt üretmek için araçları kaç kez kullanmaları gerektiğine karar verebilir. Bu, ara sonuçlara dayalı olarak birden fazla araç seçimi ve yürütme turu içerebilir.
2. **Önceden Tanımlanmış Araçlar**: Yapay zeka platformları, geliştiricilerin hazır olarak kullanabileceği Python yorumlayıcıları, web arama araçları veya yaygın yardımcı işlevler gibi önceden tanımlanmış bir dizi araç sağlayabilir.
3. **Sorunsuz Entegrasyon**: Araç kullanımı yaygınlaştıkça, yapay zeka platformları ile popüler geliştirme çerçeveleri arasında daha iyi entegrasyon bekleyebiliriz, bu da geliştiricilerin uygulamalarına araç kullanımını dahil etmesini kolaylaştıracaktır.

Araç kullanımı, geliştiricilerin yapay zeka destekli uygulamalarda BDM'lerin tam potansiyelinden yararlanmalarını sağlayan güçlü bir tekniktir. BDM'leri harici araçlara ve kaynaklara bağlayarak, kullanıcı ihtiyaçlarına uyum sağlayabilen ve değerli içgörüler ve eylemler sağlayabilen daha dinamik, akıllı ve bağlama duyarlı sistemler oluşturabilirsiniz.

Araç kullanımı muazzam olanaklar sunarken, potansiyel zorlukların ve hususların farkında olmak önemlidir. Önemli bir husus, araç etkileşimlerinin karmaşıklığını yönetmek ve genel sistemin istikrarını ve güvenilirliğini sağlamaktır. Araç çağrılarının başarısız olabileceği, beklenmeyen sonuçlar döndürebileceği veya performans etkileri olabileceği senaryoları ele almanız gerekir. Ek olarak, araçların yetkisiz veya kötü niyetli kullanımını önlemek için güvenlik ve erişim kontrolü önlemlerini düşünmelisiniz. Yapay zeka destekli uygulamanızın bütünlüğünü ve performansını korumak için uygun hata işleme, günlükleme ve izleme mekanizmaları çok önemlidir.

Kendi projelerinizde araç kullanımının olanaklarını keşfederken, net hedeflerle başlamayı, iyi yapılandırılmış araç tanımlamaları tasarlamayı ve geri bildirimler ile sonuçlara dayalı yineleme yapmayı unutmayın. Doğru yaklaşım ve düşünce yapısıyla, araç kullanımı yapay zeka destekli uygulamalarınızda yeni yenilik ve değer seviyelerinin önünü açabilir.

Akış İşleme

HTTP üzerinden akan veri, diğer adıyla sunucu gönderimli olaylar (SSE), sunucunun istemcinin açıkça talep etmesine gerek kalmadan, veriler kullanılabilir hale geldikçe sürekli olarak istemciye gönderdiği bir mekanizmadır. Yapay zeka yanıtı aşamalı olarak oluşturulduğundan, yapay zekanın çıktısını oluştukça görüntüleyerek duyarlı bir kullanıcı deneyimi sağlamak mantıklıdır. Ve aslında bildiğim tüm yapay zeka sağlayıcılarının API'leri, tamamlama uç noktalarında bir seçenek olarak akan yanıtlar sunmaktadır.

Bu bölümün kitapta tam da Araçları Kullanma bölümünden sonra yer almasının nedeni, araç kullanımını canlı yapay zeka yanıtlarıyla birleştirmenin ne kadar güçlü olabileceğidir. Bu sayede yapay zekanın kullanıcı girdisini işleyebildiği, kendi takdirine göre çeşitli araç ve fonksiyonları kullanabildiği ve gerçek zamanlı yanıtlar sağlayabildiği dinamik ve etkileşimli deneyimler mümkün olmaktadır.

Bu kesintisiz etkileşimi sağlamak için, yapay zeka tarafından çağrılan araç fonksiyonlarını ve düz metin çıktısını son kullanıcıya iletebilen akış işleyicileri yazmanız gerekir. Bir araç fonksiyonunu işledikten sonra döngüye devam etme ihtiyacı, işe ilginç bir zorluk katmaktadır.

ReplyStream'in Uygulanması

Akış işlemenin nasıl uygulanabileceğini göstermek için, bu bölümde Olympia'da kullanılan `ReplyStream` sınıfının basitleştirilmiş bir versiyonunu derinlemesine inceleyeceğiz. Bu sınıfın örnekleri, ruby-openai ve openrouter gibi yapay zeka istemci kütüphanelerinde `stream` parametresi olarak geçirilebilir.

İşte Olympia'nın `PromptSubscriber`'ında `ReplyStream`'i kullanma şeklim; bu abonelik, Wisper aracılığıyla yeni kullanıcı mesajlarının oluşturulmasını dinler.

```
1  class PromptSubscriber
2    include Raix::ChatCompletion
3    include Raix::PromptDeclarations
4
5    # many other declarations omitted...
6
7    prompt text: -> { user_message.content },
8           stream: -> { ReplyStream.new(self) },
9           until: -> { bot_message.complete? }
10
11   def message_created(message) # invoked by Wisper
12     return unless message.role.user? && message.content?
13
14     # rest of the implementation omitted...
```

`ReplyStream` sınıfı, onu örneklendiren prompt abonesine olan bir `context` referansının yanı sıra, alınan veriyi depolamak için bir tampon ve akış işleme sırasında çağrılan fonksiyon isimlerini ve argümanlarını takip etmek için diziler içeren örnek değişkenlere sahiptir.

```ruby
class ReplyStream
  attr_accessor :buffer, :f_name, :f_arguments, :context

  delegate :bot_message, :dispatch, to: :context

  def initialize(context)
    self.context = context
    self.buffer = []
    self.f_name = []
    self.f_arguments = []
  end

  def call(chunk, bytesize = nil)
    # ...
  end

  # ...
end
```

`initialize` metodu, `ReplyStream` örneğinin başlangıç durumunu ayarlar, tamponu, bağlamı ve diğer değişkenleri başlatır.

`call` metodu, akan verileri işlemek için ana giriş noktasıdır. Bir veri parçasını (karma olarak temsil edilir) ve isteğe bağlı bir `bytesize` parametresini alır; örneğimizde bu parametre kullanılmamaktadır. Bu metodun içinde, sınıf, alınan veri parçasının yapısına bağlı olarak farklı senaryoları işlemek için örüntü eşleme kullanır.

Veri parçası üzerinde `deep_symbolize_keys` çağrısı yapmak, dizgiler yerine sembollerle çalışmamızı sağlayarak örüntü eşlemeyi daha zarif hale getirir.

```
def call(chunk, _bytesize)
  case chunk.deep_symbolize_keys

  in { # match function name
    choices: [
      {
        delta: {
          tool_calls: [
            { index: index, function: {name: name} }
          ]
        }
      }
    ] }

    f_name[index] = name
```

Eşleştirdiğimiz ilk desen, bir araç çağrısı ve ona bağlı fonksiyon adıdır. Böyle bir çağrı tespit ettiğimizde, bunu `f_name` dizisine yerleştiriyoruz. Fonksiyon isimlerini dizinli bir dizide saklıyoruz, çünkü model paralel fonksiyon çağrısı yapabilme özelliğine sahip ve aynı anda birden fazla fonksiyonu çalıştırmak üzere gönderebiliyor.

> Paralel fonksiyon çağrısı, bir yapay zeka modelinin birden fazla fonksiyon çağrısını birlikte gerçekleştirebilme ve bu fonksiyon çağrılarının etkilerinin ve sonuçlarının paralel olarak çözümlenmesine izin verme yeteneğidir. Bu özellikle fonksiyonların uzun sürdüğü durumlarda faydalıdır ve API ile yapılan gidiş gelişleri azaltır, bu da önemli miktarda token tasarrufu sağlayabilir.

Sonrasında fonksiyon çağrılarına karşılık gelen argümanları eşleştirmemiz gerekiyor.

```
  in { # match arguments
    choices: [
      {
        delta: {
          tool_calls: [
            {
              index: index, function: {arguments: argument }
            }
          ]
        }
      }
    ]}

    f_arguments[index] ||= "" # initialize if not already
    f_arguments[index] << argument
```

Fonksiyon isimlerini ele aldığımız şekilde benzer şekilde, argümanları indeksli bir diziye yerleştiriyoruz.

Sırada, sunucudan tek bir belirteç şeklinde gelecek ve `new_content` değişkenine atanacak olan normal kullanıcıya yönelik mesajları inceliyoruz. Ayrıca `finish_reason` değerini de takip etmemiz gerekiyor. Bu değer, çıktı dizisinin son parçasına kadar `nil` olacak.

```
  in {
    choices: [
      { delta: {content: new_content}, finish_reason: finish_reason }
    ]}

    # you could transmit every chunk to the user here...
    buffer << new_content.to_s

    if finish_reason.present?
      finalize
    elsif new_content.to_s.match?(/\n\n/)
      send_to_client # ...or buffer and transmit once per paragraph
    end
```

Önemli olarak, AI model sağlayıcısı tarafından gönderilen hata mesajlarını işlemek

için bir desen eşleştirme ifadesi ekliyoruz. Yerel geliştirme ortamlarında bir istisna fırlatırken, prodüksiyonda hatayı günlüğe kaydedip sonlandırıyoruz.

```
  in { error: { message: } }
    if Rails.env.local?
      raise message
    else
      Honeybadger.notify("AI Error: #{message}")
      finalize
    end
```

Case yapısındaki son else ifadesi, önceki örüntülerden hiçbiri eşleşmediğinde çalışır. Bu sadece YZ modeli bize tanınmayan parçalar göndermeye başladığında bundan haberdar olmamızı sağlayan bir güvenlik önlemidir.

```
  else
    Honeybadger.notify("Unrecognized Chunk: #{chunk}")
  end
end
```

`send_to_client` metotu, arabelleklenen içeriği istemciye göndermekten sorumludur. Arabelleğin boş olmadığını kontrol eder, bot mesajı içeriğini günceller, bot mesajını oluşturur ve veri kalıcılığını sağlamak için içeriği veritabanına kaydeder.

```
def send_to_client
  # no need to process pure whitespace
  return if buffer.join.squish.blank?

  # set the buffer content on the bot message
  content = buffer.join
  bot_message.content = content

  # save to database so that we never lose data
  # even if the stream doesn't terminate correctly
  bot_message.update_column(:content, content)

  # update content via websocket
  ConversationRenderer.update(bot_message)
end
```

`finalize` metodu, akış işleme tamamlandığında çağrılır. Akış sırasında alınan fonksiyon çağrıları varsa bunları işler, bot mesajını son içerik ve diğer ilgili bilgilerle günceller ve fonksiyon çağrı geçmişini sıfırlar.

```
def finalize
  if f_name.any?
    f_name.each_with_index do |name, index|
      # takes care of calling the function wherever it's implemented
      dispatch(name:, arguments: JSON.parse(f_arguments[index]))
    end

    # reset the function call history
    f_name.clear
    f_arguments.clear
  else
    content = buffer.join.presence
    bot_message.update!(content:, complete: true)
    ConversationRenderer.update(bot_message)
  end
end
```

Eğer model bir fonksiyon çağırmaya karar verirse, bu fonksiyon çağrısını (isim ve argümanlar) öyle bir şekilde "sevk etmeniz" gerekir ki, çağrı gerçekleşsin ve konuşma kaydına `function_call` ve `function_result` mesajları eklensin.

Deneyimlerime göre, fonksiyon mesajlarının oluşturulmasını araç implementasyonlarına güvenmek yerine kod tabanınızın tek bir yerinde ele almak daha iyidir. Bu yaklaşım hem daha temizdir, hem de çok önemli pratik bir nedeni vardır: eğer yapay zeka modeli bir fonksiyon çağırır ve döngüye girdiğinizde kayıtta ilgili çağrı ve sonuç mesajlarını görmezse, *aynı fonksiyonu tekrar çağıracaktır*. Bu potansiyel olarak sonsuza kadar devam edebilir. Unutmayın ki yapay zeka tamamen durumsuzdur, dolayısıyla bu fonksiyon çağrılarını ona geri yansıtmazsanız, onlar hiç gerçekleşmemiş sayılır.

```ruby
# PromptSubscriber#dispatch

def dispatch(name:, arguments:)
  # adds a function_call message to the conversation transcript
  # plus dispatches to tool and returns result
  conversation.function_call!(name, arguments).then do |result|
    # add function result message to the transcript
    conversation.function_result!(name, result)
  end
end
```

Fonksiyon çağrı geçmişini gönderdikten sonra temizlemek, çağrının ve sonuçların dökümünüze geçtiğinden emin olmak kadar önemlidir; böylece döngü her çalıştığında aynı fonksiyonları tekrar tekrar çağırmaya devam etmezsiniz.

"Konuşma Döngüsü"

`PromptSubscriber` sınıfında, konuşma döngüsünün davranışını tanımlamak için `PromptDeclarations` modülünden `prompt` metodunu kullanırız. `until` parametresi `-> { bot_message.complete? }` olarak ayarlanmıştır; bu, döngünün `bot_message` tamamlandı olarak işaretlenene kadar devam edeceği anlamına gelir.

```ruby
prompt text: -> { user_message.content },
       stream: -> { ReplyStream.new(self) },
       until: -> { bot_message.complete? }
```

Peki `bot_message` ne zaman tamamlandı olarak işaretleniyor? Unuttuysanız, `finalize` metodunun 13. satırına geri dönün.

Tüm akış işleme mantığını gözden geçirelim.

1. `PromptSubscriber`, Wisper yayınla/abone ol sistemi aracılığıyla son kullanıcı yeni bir istem oluşturduğunda her seferinde çağrılan `message_created` metodu üzerinden yeni bir kullanıcı mesajı alır.
2. `prompt` sınıf metodu, `PromptSubscriber` için sohbet tamamlama mantığının davranışını bildirimsel olarak tanımlar. Yapay zeka modeli, kullanıcının mesaj içeriği, akış parametresi olarak yeni bir `ReplyStream` örneği ve belirtilen döngü koşulu ile bir sohbet tamamlama işlemi gerçekleştirecektir.
3. Yapay zeka modeli istemi işler ve yanıt üretmeye başlar. Yanıt akış halinde iletilirken, `ReplyStream` örneğinin `call` metodu her veri parçası için çağrılır.
4. Yapay zeka modeli bir araç fonksiyonunu çağırmaya karar verirse, fonksiyon adı ve argümanları parçadan çıkarılır ve sırasıyla `f_name` ve `f_arguments` dizilerine kaydedilir.
5. Yapay zeka modeli kullanıcıya yönelik içerik üretirse, bu içerik tamponlanır ve `send_to_client` metodu aracılığıyla istemciye gönderilir.
6. Akış işleme tamamlandığında, `finalize` metodu çağrılır. Akış sırasında herhangi bir araç fonksiyonu çağrıldıysa, bunlar `PromptSubscriber`'ın `dispatch` metodu kullanılarak gönderilir.
7. `dispatch` metodu, konuşma kaydına bir `function_call` mesajı ekler, ilgili araç fonksiyonunu çalıştırır ve fonksiyon çağrısının sonucuyla birlikte kayda bir `function_result` mesajı ekler.
8. Araç fonksiyonlarının gönderilmesinden sonra, sonraki döngülerde yinelenen fonksiyon çağrılarını önlemek için fonksiyon çağrı geçmişi temizlenir.
9. Eğer hiçbir araç fonksiyonu çağrılmadıysa, `finalize` metodu `bot_message`'ı son içerikle günceller, tamamlandı olarak işaretler ve güncellenmiş mesajı istemciye gönderir.
10. Döngü koşulu `-> { bot_message.complete? }` değerlendirilir. Eğer `bot_message` tamamlandı olarak işaretlenmediyse, döngü devam eder ve orijinal istem güncellenmiş konuşma kaydıyla tekrar gönderilir.

11. Yapay zeka modeli yanıt üretmeyi tamamlayana ve başka araç fonksiyonlarının çalıştırılması gerekmeyene kadar, yani `bot_message` tamamlandı olarak işaretlenene kadar 3-10 adımları tekrarlanır.

Bu konuşma döngüsünü uygulayarak, yapay zeka modelinin uygulama ile karşılıklı etkileşime girmesini, gerektiğinde araç fonksiyonlarını çalıştırmasını ve konuşma doğal bir sonuca ulaşana kadar kullanıcıya yönelik yanıtlar üretmesini sağlarsınız.

Akış işleme ve konuşma döngüsünün birleşimi, yapay zeka modelinin kullanıcı girdisini işleyebildiği, çeşitli araçları ve fonksiyonları kullanabildiği ve gelişen konuşma bağlamına dayalı olarak gerçek zamanlı yanıtlar sağlayabildiği dinamik ve etkileşimli yapay zeka destekli deneyimlere olanak tanır.

Otomatik Devam

Yapay zeka çıktı sınırlamalarının farkında olmak önemlidir. Çoğu modelin, `max_tokens` parametresi tarafından belirlenen, tek bir yanıtta üretebilecekleri maksimum belirteç sayısı vardır. Yapay zeka modeli yanıt üretirken bu sınıra ulaşırsa, aniden durur ve çıktının kesildiğini belirtir.

Yapay zeka platform API'sinden gelen akış yanıtında, parçadaki `finish_reason` değişkenini inceleyerek bu durumu tespit edebilirsiniz. Eğer `finish_reason` "length" olarak ayarlanmışsa (veya modele özgü başka bir anahtar değerse), bu, modelin üretim sırasında maksimum belirteç sınırına ulaştığı ve çıktının yarıda kesildiği anlamına gelir.

Bu senaryoyu zarif bir şekilde ele almanın ve sorunsuz bir kullanıcı deneyimi sağlamanın bir yolu, akış işleme mantığınıza otomatik devam mekanizması eklemektir. Uzunlukla ilgili bitiş nedenlerini eşleştiren bir kalıp ekleyerek, döngüye girmeyi ve çıktıyı kaldığı yerden otomatik olarak devam ettirmeyi seçebilirsiniz.

İşte `ReplyStream` sınıfındaki `call` metodunu otomatik devamı destekleyecek şekilde nasıl değiştirebileceğinize dair özellikle basitleştirilmiş bir örnek:

```
LENGTH_STOPS = %w[length MAX_TOKENS]

def call(chunk, _bytesize)
  case chunk.deep_symbolize_keys
    # ...

    in {
      choices: [
        { delta: {content: new_content},
          finish_reason: finish_reason } ] }

      buffer << new_content.to_s

      if finish_reason.blank?
        send_to_client if new_content.to_s.match?(/\n\n/)
      elsif LENGTH_STOPS.include?(finish_reason)
        continue_cutoff
      else
        finalize
      end

    # ...
  end
end

private

def continue_cutoff
  conversation.bot_message!(buffer.join, visible: false)
  conversation.user_message!("please continue", visible: false)
  bot_message.update_column(:created_at, Time.current)
end
```

Bu değiştirilmiş versiyonda, `finish_reason` kesik çıktıyı gösterdiğinde, akışı sonlandırmak yerine, dökümü sonlandırmadan bir çift mesaj ekliyor, orijinal kullanıcıya yönelik yanıt mesajını `created_at` özniteliğini güncelleyerek dökümün "altına" taşıyor ve sonra döngünün gerçekleşmesine izin veriyoruz, böylece yapay zeka kaldığı yerden üretmeye devam ediyor.

Yapay zeka tamamlama uç noktasının durumsuz olduğunu unutmayın. Sadece

döküm aracılığıyla ona söylediklerinizi "bilir". Bu durumda, yapay zekaya kesildiğini bildirdiğimiz yöntem, döküme (son kullanıcı için) "görünmez" mesajlar eklemektir. Ancak bunun kasıtlı olarak basitleştirilmiş bir örnek olduğunu unutmayın. Gerçek bir uygulama, belirteçleri boşa harcamamak ve/veya yapay zekayı dökümdeki tekrarlanan asistan mesajlarıyla karıştırmamak için daha fazla döküm yönetimi yapmalıdır.

Otomatik devam etmenin gerçek bir uygulaması aynı zamanda kontrolsüz döngüleri önlemek için devre kesici mantığı içermelidir. Bunun nedeni, belirli kullanıcı istemlerine ve düşük `max_tokens` ayarlarına bağlı olarak, yapay zekanın kullanıcıya yönelik çıktıyı sonsuz şekilde döngüye sokabilmesidir.

> Her döngünün ayrı bir istek gerektirdiğini ve her isteğin tüm dökümünüzü tekrar tükettiğini unutmayın. Uygulamanızda otomatik devam etmeyi uygulamaya karar verirken kullanıcı deneyimi ve API kullanımı arasındaki dengeyi mutlaka değerlendirmelisiniz. Özellikle premium ticari modeller kullanırken, otomatik devam etme tehlikeli derecede pahalı olabilir.

Sonuç

Akış işleme, araç kullanımını canlı yapay zeka yanıtlarıyla birleştiren yapay zeka destekli uygulamaların kritik bir yönüdür. Yapay zeka platform API'lerinden gelen akış verilerini verimli bir şekilde işleyerek, kesintisiz ve etkileşimli bir kullanıcı deneyimi sağlayabilir, büyük yanıtları işleyebilir, kaynak kullanımını optimize edebilir ve hataları zarif bir şekilde yönetebilirsiniz.

Sağlanan `Conversation::ReplyStream` sınıfı, örüntü eşleme ve olay güdümlü mimari kullanarak bir Ruby uygulamasında akış işlemenin nasıl uygulanabileceğini göstermektedir. Akış işleme tekniklerini anlayarak ve bunlardan yararlanarak,

uygulamalarınızda yapay zeka entegrasyonunun tüm potansiyelini ortaya çıkarabilir ve güçlü ve ilgi çekici kullanıcı deneyimleri sunabilirsiniz.

Kendini Onaran Veri

Kendini onaran veri, büyük dil modellerinin (BDM'ler) yeteneklerinden yararlanarak uygulamalarda veri bütünlüğünü, tutarlılığını ve kalitesini sağlamaya yönelik güçlü bir yaklaşımdır. Bu örüntü kategorisi, geliştiricilerin yükünü azaltmak ve yüksek düzeyde veri güvenilirliğini korumak için yapay zekayı kullanarak veri anomalilerini, tutarsızlıklarını veya hatalarını otomatik olarak tespit etme, teşhis etme ve düzeltme fikrine odaklanır.

Özünde, kendini onaran veri örüntüleri, verinin herhangi bir uygulamanın can damarı olduğunu ve doğruluğunu ve bütünlüğünü sağlamanın, uygulamanın düzgün çalışması ve kullanıcı deneyimi için çok önemli olduğunu kabul eder. Ancak, veri kalitesini yönetmek ve sürdürmek, özellikle uygulamalar büyüdükçe ve karmaşıklaştıkça zorlu ve zaman alıcı bir görev olabilir. İşte yapay zekanın gücü burada devreye girer.

Kendini onaran veri örüntülerinde, yapay zeka çalışanları uygulamanızın verisini sürekli olarak izler ve analiz eder. Bu modeller, veri içindeki örüntüleri, ilişkileri ve

anomalileri anlama ve yorumlama yeteneğine sahiptir. Doğal dil işleme ve anlama yeteneklerinden yararlanarak, verideki potansiyel sorunları veya tutarsızlıkları belirleyebilir ve bunları düzeltmek için uygun önlemleri alabilirler.

Kendini onaran veri süreci genellikle birkaç önemli adımı içerir:

1. **Veri İzleme**: Yapay zeka çalışanları, uygulamanın veri akışlarını, veritabanlarını veya depolama sistemlerini sürekli olarak izleyerek anomali, tutarsızlık veya hata belirtilerini arar. Alternatif olarak, bir istisna durumunda yapay zeka bileşenini etkinleştirebilirsiniz.
2. **Anomali Tespiti**: Bir sorun tespit edildiğinde, yapay zeka çalışanı sorunun özel niteliğini ve kapsamını belirlemek için veriyi detaylı olarak analiz eder. Bu, eksik değerleri, tutarsız biçimleri veya önceden tanımlanmış kuralları veya kısıtlamaları ihlal eden verileri tespit etmeyi içerebilir.
3. **Teşhis ve Düzeltme**: Sorun belirlendikten sonra, yapay zeka çalışanı uygun eylem planını belirlemek için veri alanındaki bilgi ve anlayışını kullanır. Bu, veriyi otomatik olarak düzeltmeyi, eksik değerleri doldurmayı veya gerekirse insan müdahalesi için sorunu işaretlemeyi içerebilir.
4. **Sürekli Öğrenme (kullanım senaryosuna bağlı olarak isteğe bağlı)**: Yapay zeka çalışanınız çeşitli veri sorunlarıyla karşılaştıkça ve bunları çözdükçe, ne olduğunu ve nasıl yanıt verdiğini açıklayan çıktılar üretebilir. Bu üstveri, size (ve belki de ince ayar yoluyla temel modele) veri anomalilerini belirleme ve çözme konusunda zaman içinde daha etkili ve verimli olma imkanı sağlayan öğrenme süreçlerine beslenebilir.

Veri sorunlarını otomatik olarak tespit edip düzelterek, uygulamanızın yüksek kaliteli, güvenilir verilerle çalışmasını sağlayabilirsiniz. Bu, uygulamanın işlevselliğini veya kullanıcı deneyimini etkileyebilecek hataların, tutarsızlıkların veya veriyle ilgili hataların riskini azaltır.

Yapay zeka çalışanları veri izleme ve düzeltme görevini üstlendikten sonra, çabalarınızı uygulamanın diğer kritik yönlerine odaklayabilirsiniz. Bu, manuel veri temizleme ve

bakımına harcanacak zaman ve kaynakları tasarruf eder. Aslında, uygulamalarınız büyüdükçe ve karmaşıklaştıkça, veri kalitesini manuel olarak yönetmek giderek zorlaşır. "Kendini Onaran Veri" örüntüleri, büyük veri hacimlerini işlemek ve sorunları gerçek zamanlı olarak tespit etmek için yapay zekanın gücünden yararlanarak etkili bir şekilde ölçeklenir.

Doğaları gereği, yapay zeka modelleri çok az denetim ile veya hiç denetim olmadan zamanla değişen veri örüntülerine, şemalarına veya gereksinimlerine uyum sağlayabilir. Yönergeleri, özellikle hedeflenen sonuçlar açısından yeterli rehberlik sağladığı sürece, uygulamanız kapsamlı manuel müdahale veya kod değişiklikleri gerektirmeden evrimleşebilir ve yeni veri senaryolarını ele alabilir.

Kendini onaran veri örüntüleri, tartıştığımız "Çalışanların Çokluğu" gibi diğer örüntü kategorileriyle iyi uyum sağlar. Kendini onaran veri yeteneği, özellikle veri kalitesini ve bütünlüğünü sağlamaya odaklanan özelleşmiş bir tür çalışan olarak görülebilir. Bu tür çalışan, uygulamanın işlevselliğinin farklı yönlerine katkıda bulunan diğer yapay zeka çalışanlarıyla birlikte çalışır.

Kendini onaran veri örüntülerini pratikte uygulamak, yapay zeka modellerinin uygulama mimarisine dikkatli bir şekilde tasarlanmasını ve entegrasyonunu gerektirir. Veri kaybı ve bozulma riskleri nedeniyle, bu tekniği nasıl kullanacağınıza dair net yönergeler belirlemelisiniz. Ayrıca performans, ölçeklenebilirlik ve veri güvenliği gibi faktörleri de göz önünde bulundurmalısınız.

Pratik Vaka Çalışması: Bozuk JSON'ı Düzeltme

Kendini onaran veriyi kullanmanın en pratik ve uygun yollarından biri aynı zamanda açıklaması da çok basittir: bozuk JSON'ı düzeltmek.

Bu teknik, BDM'ler tarafından üretilen bozuk JSON gibi kusurlu veya tutarsız verilerle başa çıkma gibi yaygın zorluklara uygulanabilir ve bu sorunları otomatik olarak tespit etme ve düzeltme için bir yaklaşım sunar.

Olympia'da sıklıkla LLM'lerin tam olarak geçerli olmayan JSON verileri ürettiği senaryolarla karşılaşıyorum. Bu durum, LLM'nin asıl JSON kodundan önce veya sonra yorum eklemesi ya da eksik virgül veya kaçış karakteri kullanılmamış çift tırnak gibi sözdizimi hataları oluşturması gibi çeşitli nedenlerle meydana gelebilir. Bu sorunlar ayrıştırma hatalarına yol açabilir ve uygulamanın işlevselliğinde aksaklıklara neden olabilir.

Bu sorunu çözmek için, JsonFixer sınıfı şeklinde pratik bir çözüm geliştirdim. Bu sınıf, "Kendini Onaran Veri" desenini somutlaştırarak bozuk JSON'ı girdi olarak alır ve mümkün olduğunca fazla bilgi ve amacı koruyarak düzeltmek için bir LLM'den yararlanır.

```
class JsonFixer
  include Raix::ChatCompletion

  def call(bad_json, error_message)
    raise "No data provided" if bad_json.blank? || error_message.blank?

    transcript << {
        system: "Consider user-provided JSON that generated a parse
                 exception. Do your best to fix it while preserving the
                 original content and intent as much as possible." }
    transcript << { user: bad_json }
    transcript << { assistant: "What is the error message?"}
    transcript << { user: error_message }
    transcript << { assistant: "Here is the corrected JSON\n```json\n" }

    self.stop = ["```"]

    chat_completion(json: true)
  end

  def model
    "mistralai/mixtral-8x7b-instruct:nitro"
```

```
23    end
24  end
```

`JsonFixer`'ın AI yanıtlarını yönlendirmek için Ventriloquist'i nasıl kullandığına dikkat edin.

Kendi kendini onaran JSON verilerinin işlem süreci şu şekildedir:

1. **JSON Oluşturma**: Belirli istemler veya gereksinimlere dayalı olarak JSON verisi oluşturmak için bir LLM kullanılır. Ancak, LLM'lerin doğası gereği, oluşturulan JSON her zaman tam olarak geçerli olmayabilir. Elbette JSON ayrıştırıcı, geçersiz bir JSON verdiğinizde bir `ParserError` hatası fırlatacaktır.

```
1  begin
2    JSON.parse(llm_generated_json)
3  rescue JSON::ParserError => e
4    JsonFixer.new.call(llm_generated_json, e.message)
5  end
```

Exception mesajının da `JSONFixer` çağrısına iletildiğine dikkat edin, böylece veriyle ilgili neyin yanlış olduğunu tam olarak varsaymak zorunda kalmaz, özellikle ayrıştırıcı genellikle size tam olarak neyin yanlış olduğunu söyleyecektir.

2. **BDM-tabanlı Düzeltme**: `JSONFixer` sınıfı, bozuk JSON'ı bir BDM'ye (Büyük Dil Modeli) geri gönderir ve orijinal bilgi ve amacı mümkün olduğunca koruyarak JSON'ı düzeltmek için özel bir prompt veya talimat ekler. Büyük miktarda veri üzerinde eğitilmiş ve JSON sözdizimini anlayan BDM, hataları düzeltmeye ve geçerli bir JSON dizesi oluşturmaya çalışır. BDM'nin çıktısını sınırlamak için Yanıt Sınırlama kullanılır ve bu tür görevler için özellikle iyi olan Mixtral 8x7B'yi yapay zeka modeli olarak seçeriz.

3. **Doğrulama ve Entegrasyon**: BDM tarafından döndürülen düzeltilmiş JSON dizesi, `chat_completion(json: true)` çağrısı yaptığı için `JSONFixer` sınıfının kendisi tarafından ayrıştırılır. Düzeltilmiş JSON doğrulamayı geçerse, uygulamanın iş akışına tekrar entegre edilir ve uygulamanın veriyi sorunsuz bir şekilde işlemeye devam etmesini sağlar. Bozuk JSON "iyileştirilmiş" olur.

Kendi `JSONFixer` uygulamalarımı defalarca yazmış ve yeniden yazmış olmama rağmen, tüm bu versiyonlara harcanan toplam zamanın bir veya iki saatten fazla olduğundan şüpheliyim.

Amacın korunmasının, herhangi bir kendi kendini iyileştiren veri deseninin kilit unsuru olduğunu unutmayın. BDM-tabanlı düzeltme süreci, oluşturulan JSON'ın orijinal bilgi ve amacını mümkün olduğunca korumayı hedefler. Bu, düzeltilmiş JSON'ın anlamsal önemini korumasını ve uygulama bağlamında etkili bir şekilde kullanılabilmesini sağlar.

Olympia'daki "Kendi Kendini İyileştiren Veri" yaklaşımının bu pratik uygulaması, yapay zekanın, özellikle BDM'lerin, gerçek dünya veri zorluklarını çözmek için nasıl kullanılabileceğini açıkça göstermektedir. Sağlam ve verimli uygulamalar oluşturmak için geleneksel programlama tekniklerini yapay zeka yetenekleriyle birleştirmenin gücünü sergiler.

Postel Yasası ve "Kendi Kendini İyileştiren Veri" Deseni

JSONFixer sınıfı örneğinde görüldüğü gibi "Kendi Kendini İyileştiren Veri", Postel Yasası olarak da bilinen Sağlamlık İlkesi ile uyumludur. Postel Yasası şöyle der:

"Yaptığınız şeyde muhafazakar, başkalarından kabul ettiğiniz şeylerde liberal olun."

İnternetin ilk dönemlerinin öncülerinden Jon Postel tarafından ortaya konan bu ilke, çıktı gönderirken belirtilen protokollere sıkı sıkıya bağlı kalırken, çeşitli veya hatta

hafifçe hatalı girdilere karşı toleranslı sistemler kurmanın önemini vurgular.

"Kendi Kendini İyileştiren Veri" bağlamında, JSONFixer sınıfı, BDM'ler tarafından üretilen bozuk veya kusurlu JSON verilerini kabul etmede liberal davranarak Postel Yasası'nı somutlaştırır. Beklenen formata tam olarak uymayan JSON ile karşılaştığında hemen reddetmez veya başarısız olmaz. Bunun yerine, toleranslı bir yaklaşım benimser ve BDM'lerin gücünü kullanarak JSON'ı düzeltmeye çalışır.

Kusurlu JSON'ı kabul etmede liberal davranarak, JSONFixer sınıfı sağlamlık ve esneklik gösterir. Gerçek dünyada verilerin çeşitli biçimlerde geldiğini ve her zaman katı spesifikasyonlara uymayabileceğini kabul eder. Bu sapmaları zarif bir şekilde ele alıp düzelterek, sınıf uygulamanın kusurlu veri varlığında bile sorunsuz çalışmaya devam etmesini sağlar.

Öte yandan, JSONFixer sınıfı çıktı söz konusu olduğunda Postel Yasası'nın muhafazakar yönüne de uyar. BDM'leri kullanarak JSON'ı düzelttikten sonra, sınıf düzeltilmiş JSON'ı doğrulayarak beklenen formata sıkı sıkıya uymasını sağlar. Uygulamanın diğer bölümlerine aktarmadan önce verinin bütünlüğünü ve doğruluğunu korur. Bu muhafazakar yaklaşım, JSONFixer sınıfının çıktısının güvenilir ve tutarlı olmasını garanti eder, birlikte çalışabilirliği teşvik eder ve hataların yayılmasını önler.

Jon Postel Hakkında İlginç Bilgiler:

- Jon Postel (1943-1998), İnternetin gelişiminde çok önemli bir rol oynayan Amerikalı bir bilgisayar bilimcisiydi. Temel protokollere ve standartlara yaptığı önemli katkılardan dolayı "İnternetin Tanrısı" olarak bilinirdi.
- Postel, İnternet hakkındaki teknik ve organizasyonel notların bir serisi olan Request for Comments (RFC) belge serisinin editörüydü. TCP, IP ve SMTP gibi temel protokoller de dahil olmak üzere 200'den fazla RFC'nin yazarı veya ortak yazarıydı.
- Teknik katkılarının yanı sıra, mütevazı ve işbirlikçi yaklaşımıyla tanınırdı. Sağlam ve birlikte çalışabilir bir ağ oluşturmak için fikir birliğine varmanın

ve birlikte çalışmanın önemine inanırdı.
- Postel, 1977'den 1998'deki zamansız ölümüne kadar Güney Kaliforniya Üniversitesi (USC) Bilgi Bilimleri Enstitüsü'nün (ISI) Bilgisayar Ağları Bölümü'nün Direktörü olarak görev yaptı.
- Muazzam katkılarının tanınması amacıyla, 1998'de genellikle "Bilgisayar Bilimlerinin Nobel Ödülü" olarak anılan prestijli Turing Ödülü'ne ölümünden sonra layık görüldü.

`JSONFixer` sınıfı, Postel'in kariyeri boyunca savunduğu temel değerler olan sağlamlık, esneklik ve birlikte çalışabilirliği destekler. Protokollere sıkı bağlılığı korurken kusurları tolere eden sistemler inşa ederek, gerçek dünyadaki zorluklarla karşılaştığında daha dayanıklı ve uyarlanabilir uygulamalar oluşturabiliriz.

Değerlendirmeler ve Karşı Göstergeler

Kendini onaran veri yaklaşımlarının uygulanabilirliği tamamen uygulamanızın işlediği veri türüne bağlıdır. Uygulamanızdaki *tüm JSON ayrıştırma hatalarını* otomatik olarak düzeltmek için `JSON.parse`'ı basitçe monkeypatch etmek istememenizin bir nedeni vardır: tüm hatalar otomatik olarak düzeltilemez veya düzeltilmemelidir.

Kendini onarma özellikle veri işleme ve yönetimi ile ilgili düzenleyici veya uyumluluk gereksinimleriyle birleştiğinde sorunlu hale gelir. Sağlık ve finans gibi bazı sektörler, veri bütünlüğü ve denetlenebilirlik konusunda o kadar katı düzenlemelere sahiptir ki, uygun gözetim veya günlük kaydı olmadan herhangi bir "kara kutu" veri düzeltmesi yapmak bu düzenlemeleri ihlal edebilir. Geliştirdiğiniz kendini onaran veri tekniklerinin geçerli yasal ve düzenleyici çerçevelerle uyumlu olmasını sağlamak çok önemlidir.

Özellikle yapay zeka modellerini içeren kendini onaran veri tekniklerinin uygulanması, uygulama performansı ve kaynak kullanımı üzerinde büyük bir etkiye sahip olabilir. Hata tespiti ve düzeltme için büyük miktarda veriyi yapay zeka modellerinden

geçirmek hesaplama açısından yoğun olabilir. Kendini onaran verinin faydaları ile ilişkili performans ve kaynak maliyetleri arasındaki dengeyi değerlendirmek önemlidir.

Bu güçlü yaklaşımı ne zaman ve nerede uygulayacağımıza karar vermede rol oynayan faktörlere daha yakından bakalım.

Veri Kritikliği

Kendini onaran veri tekniklerinin uygulanmasını değerlendirirken, işlenen verinin kritikliğini değerlendirmek çok önemlidir. Kritiklik düzeyi, verinin uygulamanız ve iş alanı bağlamındaki önemini ve hassasiyetini ifade eder.

Bazı durumlarda, veri hatalarını otomatik olarak düzeltmek uygun olmayabilir, özellikle veri yüksek derecede hassas ise veya yasal etkileri varsa. Örneğin, aşağıdaki senaryoları düşünün:

1. **Finansal İşlemler**: Bankacılık sistemleri veya ticaret platformları gibi finansal uygulamalarda, veri doğruluğu son derece önemlidir. Finansal verilerdeki küçük hatalar bile yanlış hesap bakiyeleri, yanlış yönlendirilmiş fonlar veya hatalı ticaret kararları gibi önemli sonuçlara yol açabilir. Bu durumlarda, kapsamlı doğrulama ve denetim olmadan yapılan otomatik düzeltmeler kabul edilemez riskler getirebilir.
2. **Tıbbi Kayıtlar**: Sağlık uygulamaları, son derece hassas ve gizli hasta verileriyle ilgilenir. Tıbbi kayıtlardaki yanlışlıklar, hasta güvenliği ve tedavi kararları üzerinde ciddi etkilere sahip olabilir. Kalifiye sağlık profesyonelleri tarafından uygun gözetim ve doğrulama olmadan tıbbi verileri otomatik olarak değiştirmek, düzenleyici gereksinimleri ihlal edebilir ve hasta refahını riske atabilir.
3. **Yasal Belgeler**: Sözleşmeler, anlaşmalar veya mahkeme başvuruları gibi yasal belgeleri işleyen uygulamalar, katı doğruluk ve bütünlük gerektirir. Yasal verilerdeki küçük hatalar bile önemli yasal sonuçlara yol açabilir. Bu alanda otomatik düzeltmeler uygun olmayabilir, çünkü verinin geçerliliğini ve

uygulanabilirliğini sağlamak için genellikle hukuk uzmanları tarafından manuel inceleme ve doğrulama gerekir.

Bu kritik veri senaryolarında, otomatik düzeltmelerle ilişkili riskler genellikle potansiyel faydalardan daha ağır basar. Hataları tanıtmanın veya verileri yanlış değiştirmenin sonuçları, finansal kayıplar, yasal sorumluluklar veya hatta bireylere zarar verme gibi ciddi olabilir.

Yüksek derecede kritik verilerle uğraşırken, manuel doğrulama ve onaylama süreçlerine öncelik vermek esastır. Verinin doğruluğunu ve bütünlüğünü sağlamada insan gözetimi ve uzmanlığı çok önemlidir. Otomatik kendini onaran teknikler hala potansiyel hataları veya tutarsızlıkları işaretlemek için kullanılabilir, ancak düzeltmelerle ilgili nihai karar insan muhakemesi ve onayını içermelidir.

Bununla birlikte, bir uygulamadaki tüm verilerin aynı kritiklik düzeyine sahip olmayabileceğini unutmamak önemlidir. Aynı uygulama içinde, daha az hassas olan veya hata oluştuğunda daha düşük etkiye sahip veri alt kümeleri olabilir. Bu gibi durumlarda, kendini onaran veri teknikleri bu belirli veri alt kümelerine seçici olarak uygulanabilirken, kritik veriler manuel doğrulamaya tabi olmaya devam eder.

Önemli olan, uygulamanızdaki her veri kategorisinin kritikliğini dikkatle değerlendirmek ve ilişkili riskler ve sonuçlara dayalı olarak düzeltmeleri ele almak için net yönergeler ve süreçler tanımlamaktır. Kritik (örneğin defterler, tıbbi kayıtlar) ve kritik olmayan veriler (örneğin posta adresleri, kaynak uyarıları) arasında ayrım yaparak, kendini onaran veri tekniklerinin faydalarından uygun olduğu yerde yararlanmak ve gerekli olduğu yerde sıkı kontrol ve gözetimi sürdürmek arasında bir denge kurabilirsiniz.

Sonuç olarak, kendini onaran veri tekniklerini kritik verilere uygulama kararı, alan uzmanları, hukuk danışmanları ve diğer ilgili paydaşlarla istişare içinde alınmalıdır. Uygulamanızın verileriyle ilişkili özel gereksinimleri, düzenlemeleri ve riskleri dikkate almak ve veri düzeltme stratejilerini buna göre hizalamak esastır.

Hata Şiddeti

Kendini onaran veri tekniklerini uygularken, veri hatalarının şiddetini ve etkisini değerlendirmek önemlidir. Tüm hatalar eşit değildir ve uygun eylem şekli hatanın ciddiyetine bağlı olarak değişebilir.

Küçük tutarsızlıklar veya biçimlendirme sorunları otomatik düzeltme için uygun olabilir. Örneğin, bozuk JSON'u düzeltmekle görevlendirilmiş bir kendini onaran veri işçisi, verinin anlamını veya yapısını önemli ölçüde değiştirmeden eksik virgülleri veya kaçış karakteri olmayan çift tırnakları ele alabilir. Bu tür hatalar genellikle düzeltilmesi kolaydır ve genel veri bütünlüğü üzerinde minimal etkiye sahiptir.

Bununla birlikte, verinin anlamını veya bütünlüğünü temelden değiştiren daha ciddi hatalar, farklı bir yaklaşım gerektirebilir. Bu gibi durumlarda, otomatik düzeltmeler yeterli olmayabilir ve verinin doğruluğunu ve geçerliliğini sağlamak için insan müdahalesi gerekebilir.

İşte tam bu noktada, hata ciddiyetini belirlemede yapay zekanın kendisinden yararlanma kavramı devreye girer. Yapay zeka modellerinin yeteneklerinden faydalanarak, sadece hataları düzeltmekle kalmayıp, aynı zamanda bu hataların ciddiyetini değerlendiren ve bunları nasıl ele alacağına dair bilinçli kararlar verebilen kendini onaran veri işçileri tasarlayabiliriz.

Örneğin, bir müşteri veritabanına akan verilerdeki tutarsızlıkları düzeltmekten sorumlu bir kendini onaran veri işçisini ele alalım. Bu işçi, verileri analiz etmek ve eksik veya çelişkili bilgiler gibi olası hataları tespit etmek üzere tasarlanabilir. Ancak tüm hataları otomatik olarak düzeltmek yerine, işçi ciddi hataları insan incelemesi için işaretlemesine olanak tanıyan ek araç çağrıları ile donatılabilir.

İşte bunun nasıl uygulanabileceğine dair bir örnek:

```ruby
class CustomerDataReviewer
  include Raix::ChatCompletion
  include Raix::FunctionDeclarations

  attr_accessor :customer

  function :flag_for_review, reason: { type: "string" } do |params|
    AdminNotifier.review_request(customer, params[:reason])
  end

  def initialize(customer)
    self.customer = customer
  end

  def call(customer_data)
    transcript << {
      system: "You are a customer data reviewer. Your task is to identify
        and correct inconsistencies in customer data.

        < additional instructions here... >

        If you encounter severe errors that require human review, use the
        `flag_for_review` tool to flag the data for manual intervention." }

    transcript << { user: customer.to_json }
    transcript << { assistant: "Reviewed/corrected data:\n```json\n" }

    self.stop = ["```"]

    chat_completion(json: true).then do |result|
      return if result.blank?

      customer.update(result)
    end
  end
end
```

Bu örnekte, `CustomerDataHealer` işçisi müşteri verilerindeki tutarsızlıkları belirleme ve düzeltme için tasarlanmıştır. Bir kez daha, yapılandırılmış çıktı almak için Yanıt Sınırlama ve Karın Konuşmacısı kullanıyoruz. Önemli olarak, işçinin sistem

yönergesi, ciddi hatalarla karşılaşıldığında `flag_for_review` fonksiyonunu kullanma talimatlarını içerir.

İşçi müşteri verilerini işlerken, verileri analiz eder ve tutarsızlıkları düzeltmeye çalışır. İşçi hataların ciddi olduğunu ve insan müdahalesi gerektirdiğini belirlerse, `flag_for_review` aracını kullanarak verileri işaretleyebilir ve işaretleme nedenini belirtebilir.

`chat_completion` metodu, düzeltilmiş müşteri verilerini JSON olarak ayrıştırmak için `json: true` parametresiyle çağrılır. Bir fonksiyon çağrısından sonra döngü yapma imkanı olmadığından, eğer `flag_for_review` çağrıldıysa sonuç boş olacaktır. Aksi takdirde, müşteri incelenmiş ve potansiyel olarak düzeltilmiş verilerle güncellenir.

Hata ciddiyeti değerlendirmesi ve verileri insan incelemesi için işaretleme seçeneğini dahil ederek, kendini onaran veri işçisi daha akıllı ve uyarlanabilir hale gelir. Küçük hataları otomatik olarak ele alabilirken, ciddi hataları manuel müdahale için insan uzmanlara yönlendirebilir.

Hata ciddiyetini belirleme kriterleri, alan bilgisi ve iş gereksinimlerine dayalı olarak işçinin yönergesinde tanımlanabilir. Veri bütünlüğü üzerindeki etki, veri kaybı veya bozulma olasılığı ve yanlış verilerin sonuçları gibi faktörler ciddiyet değerlendirmesinde dikkate alınabilir.

Hata ciddiyetini değerlendirmek için yapay zekayı kullanarak ve insan müdahalesi için seçenekler sunarak, kendini onaran veri teknikleri otomasyon ile veri doğruluğunu koruma arasında bir denge kurabilir. Bu yaklaşım, küçük hataların verimli bir şekilde düzeltilmesini sağlarken, ciddi hataların insan inceleyicilerinden gerekli ilgi ve uzmanlığı almasını sağlar.

Alan Karmaşıklığı

Kendini onaran veri tekniklerinin uygulanmasını düşünürken, veri alanının karmaşıklığını ve yapısını ve ilişkilerini yöneten kuralları değerlendirmek önemlidir. Alanın karmaşıklığı, otomatik veri düzeltme yaklaşımlarının etkinliğini ve uygulanabilirliğini önemli ölçüde etkileyebilir.

Kendini onaran veri teknikleri, veriler iyi tanımlanmış kalıpları ve kısıtlamaları takip ettiğinde iyi çalışır. Veri yapısının nispeten basit olduğu ve veri öğeleri arasındaki ilişkilerin açık olduğu alanlarda, otomatik düzeltmeler yüksek bir güvenle uygulanabilir. Örneğin, biçimlendirme sorunlarını düzeltme veya temel veri tipi kısıtlamalarını uygulama genellikle kendini onaran veri işçileri tarafından etkili bir şekilde ele alınabilir.

Ancak, veri alanının karmaşıklığı arttıkça, otomatik veri düzeltmeyle ilişkili zorluklar da büyür. Karmaşık iş mantığı, veri varlıkları arasında karmaşık ilişkiler veya alana özgü kurallar ve istisnalar içeren alanlarda, kendini onaran veri teknikleri her zaman nüansları yakalayamayabilir ve istenmeyen sonuçlara yol açabilir.

Karmaşık bir alan örneği düşünelim: bir finansal ticaret sistemi. Bu alanda, veriler çeşitli finansal araçları, piyasa verilerini, ticaret kurallarını ve düzenleyici gereksinimleri içerir. Farklı veri öğeleri arasındaki ilişkiler karmaşık olabilir ve veri geçerliliği ve tutarlılığını yöneten kurallar alana oldukça özgü olabilir.

Böyle karmaşık bir alanda, ticaret verilerindeki tutarsızlıkları düzeltmekle görevlendirilmiş bir kendini onaran veri işçisinin, alana özgü kuralların ve kısıtlamaların derin bir anlayışına sahip olması gerekir. Piyasa düzenlemeleri, ticaret limitleri, risk hesaplamaları ve uzlaşma prosedürleri gibi faktörleri dikkate alması gerekir. Bu bağlamda otomatik düzeltmeler, alanın tam karmaşıklığını her zaman yakalayamayabilir ve istemeden hatalar veya alana özgü kural ihlalleri oluşturabilir.

Alan karmaşıklığının zorluklarını ele almak için, kendini onaran veri teknikleri alana özgü bilgi ve kuralları yapay zeka modellerine ve işçilerine dahil ederek geliştirilebilir. Bu, aşağıdaki tekniklerle gerçekleştirilebilir:

1. **Alana Özgü Eğitim**: Kendini onaran veri için kullanılan yapay zeka modelleri, belirli alanın inceliklerini ve kurallarını yakalayan alana özgü veri setleri üzerinde yönlendirilebilir veya hatta ince ayar yapılabilir. Modelleri temsili veriler ve senaryolara maruz bırakarak, alana özgü kalıpları, kısıtlamaları ve istisnaları öğrenebilirler.

2. **Kural Tabanlı Kısıtlamalar**: Kendini onaran veri işçileri, alana özgü bilgileri kodlayan açık kural tabanlı kısıtlamalarla güçlendirilebilir. Bu kurallar alan uzmanları tarafından tanımlanabilir ve veri düzeltme sürecine entegre edilebilir. Yapay zeka modelleri daha sonra bu kuralları kararlarını yönlendirmek ve alana özgü gereksinimlere uyumu sağlamak için kullanabilir.
3. **Alan Uzmanlarıyla İşbirliği**: Karmaşık alanlarda, kendini onaran veri tekniklerinin tasarımına ve geliştirilmesine alan uzmanlarını dahil etmek çok önemlidir. Alan uzmanları, verinin incelikleri, iş kuralları ve olası uç durumlar hakkında değerli içgörüler sağlayabilir. Onların bilgileri, İnsan Destekli Döngü kalıplarını kullanarak otomatik veri düzeltmelerinin doğruluğunu ve güvenilirliğini artırmak için yapay zeka modellerine ve işçilerine dahil edilebilir.
4. **Aşamalı ve Yinelemeli Yaklaşım**: Karmaşık alanlarla uğraşırken, genellikle kendini onaran veriye aşamalı ve yinelemeli bir yaklaşım benimsemek faydalıdır. Tüm alan için bir kerede düzeltmeleri otomatikleştirmeye çalışmak yerine, kuralların ve kısıtlamaların iyi anlaşıldığı belirli alt alanlara veya veri kategorilerine odaklanın. Alan anlayışı geliştikçe ve teknikler etkili olduğunu kanıtladıkça kendini onaran tekniklerin kapsamını kademeli olarak genişletin.

Veri alanının karmaşıklığını göz önünde bulundurarak ve alana özgü bilgiyi kendini onaran veri tekniklerine dahil ederek, otomasyon ve doğruluk arasında bir denge kurabilirsiniz. Kendini onaran verinin her duruma uyan tek bir çözüm olmadığını ve yaklaşımın her alanın özel gereksinimlerine ve zorluklarına göre uyarlanması gerektiğini kabul etmek önemlidir.

Karmaşık alanlarda, kendini onaran veri tekniklerini insan uzmanlığı ve gözetimiyle birleştiren hibrit bir yaklaşım en etkili olabilir. Otomatik düzeltmeler rutin ve iyi tanımlanmış durumları ele alabilirken, karmaşık senaryolar veya istisnalar insan incelemesi ve müdahalesi için işaretlenebilir. Bu işbirlikçi yaklaşım, karmaşık veri alanlarında gerekli kontrol ve doğruluğu korurken otomasyonun faydalarının gerçekleştirilmesini sağlar.

Açıklanabilirlik ve Şeffaflık

Açıklanabilirlik, yapay zeka modellerinin verdiği kararların arkasındaki mantığı anlama ve yorumlama yeteneğini ifade ederken, şeffaflık veri düzeltme sürecine açık bir görünürlük sağlamayı içerir.

Birçok bağlamda, veri değişikliklerinin denetlenebilir ve gerekçelendirilebilir olması gerekir. İş kullanıcıları, denetçiler ve düzenleyici kurumlar dahil olmak üzere paydaşlar, belirli veri düzeltmelerinin neden yapıldığı ve yapay zeka modellerinin bu kararlara nasıl vardığı konusunda açıklamalar talep edebilir. Bu, özellikle finans, sağlık hizmetleri ve hukuki konular gibi veri doğruluğu ve bütünlüğünün önemli etkilere sahip olduğu alanlarda çok önemlidir.

Açıklanabilirlik ve şeffaflık ihtiyacını karşılamak için, kendini onaran veri teknikleri, yapay zeka modellerinin karar verme sürecine ilişkin içgörüler sağlayan mekanizmaları içermelidir. Bu, çeşitli yaklaşımlarla gerçekleştirilebilir:

1. **Düşünce Zinciri**: Modelden verilerde değişiklik yapmadan önce düşüncesini "sesli" olarak açıklamasını istemek, karar verme sürecinin daha kolay anlaşılmasını sağlayabilir ve yapılan düzeltmeler için insan tarafından okunabilir açıklamalar üretebilir. Bunun bedeli, açıklamayı yapılandırılmış veri çıktısından ayırmada biraz daha karmaşıklıktır, bu da şöyle ele alınabilir…
2. **Açıklama Üretimi**: Kendini onaran veri çalışanları, yaptıkları düzeltmeler için insan tarafından okunabilir açıklamalar üretme yeteneği ile donatılabilir. Bu, modelden karar verme sürecini *verinin kendisine entegre edilmiş* kolayca anlaşılabilir açıklamalar olarak çıktı vermesini isteyerek gerçekleştirilebilir. Örneğin, kendini onaran bir veri çalışanı, tespit ettiği spesifik veri tutarsızlıklarını, uyguladığı düzeltmeleri ve bu düzeltmelerin arkasındaki mantığı vurgulayan bir rapor oluşturabilir.
3. **Özellik Önemi**: Yapay zeka modellerine, direktiflerinin bir parçası olarak veri düzeltme sürecindeki farklı özelliklerin veya niteliklerin önemi hakkında bilgi

verilebilir. Bu direktifler de insan paydaşlara açıklanabilir. Modelin kararlarını etkileyen temel faktörleri belirleyerek, paydaşlar düzeltmelerin arkasındaki mantık hakkında içgörü kazanabilir ve bunların geçerliliğini değerlendirebilir.

4. **Günlük Tutma ve Denetleme**: Kendini onaran veri sürecinde şeffaflığı korumak için kapsamlı günlük tutma ve denetleme mekanizmalarının uygulanması çok önemlidir. Yapay zeka modelleri tarafından yapılan her veri düzeltmesi, orijinal veri, düzeltilmiş veri ve atılan spesifik adımlar dahil olmak üzere kaydedilmelidir. Bu denetim izi, geriye dönük analiz yapılmasına olanak tanır ve verilerde yapılan değişikliklerin net bir kaydını sağlar.
5. **İnsan Destekli Döngü Yaklaşımı**: İnsan destekli döngü yaklaşımının dahil edilmesi, kendini onaran veri tekniklerinin açıklanabilirliğini ve şeffaflığını artırabilir. Yapay zeka tarafından üretilen düzeltmelerin incelenmesi ve doğrulanmasında insan uzmanları dahil ederek, kuruluşlar düzeltmelerin alan bilgisi ve iş gereksinimleriyle uyumlu olmasını sağlayabilir. İnsan gözetimi ek bir hesap verebilirlik katmanı ekler ve yapay zeka modellerindeki olası önyargıların veya hataların belirlenmesine olanak tanır.
6. **Sürekli İzleme ve Değerlendirme**: Kendini onaran veri tekniklerinin performansını düzenli olarak izlemek ve değerlendirmek, şeffaflığı ve güveni korumak için çok önemlidir. Yapay zeka modellerinin doğruluğunu ve etkinliğini zaman içinde değerlendirerek, kuruluşlar herhangi bir sapma veya anomaliyi belirleyebilir ve düzeltici önlemler alabilir. Sürekli izleme, kendini onaran veri sürecinin güvenilir kalmasını ve istenen sonuçlarla uyumlu olmasını sağlamaya yardımcı olur.

Açıklanabilirlik ve şeffaflık, kendini onaran veri tekniklerinin uygulanmasında kritik öneme sahip hususlardır. Veri düzeltmeleri için net açıklamalar sağlayarak, kapsamlı denetim izleri tutarak ve insan gözetimini dahil ederek, kuruluşlar kendini onaran veri sürecine olan güveni inşa edebilir ve verilerde yapılan değişikliklerin gerekçelendirilebilir ve iş hedefleriyle uyumlu olmasını sağlayabilir.

Otomasyon faydaları ile şeffaflık ihtiyacı arasında bir denge kurmak önemlidir. Kendini onaran veri teknikleri veri kalitesini ve verimliliği önemli ölçüde iyileştirebilse de, bu veri düzeltme süreci üzerindeki görünürlük ve kontrolü kaybetme pahasına olmamalıdır. Kendini onaran veri çalışanlarını açıklanabilirlik ve şeffaflık göz önünde bulundurularak tasarlayarak, kuruluşlar yapay zekanın gücünden yararlanırken verilerde gerekli hesap verebilirlik ve güven düzeyini koruyabilir.

İstenmeyen Sonuçlar

Kendini onaran veri teknikleri veri kalitesini ve tutarlılığını iyileştirmeyi amaçlasa da, istenmeyen sonuçların ortaya çıkma potansiyeline karşı dikkatli olmak çok önemlidir. Dikkatli bir şekilde tasarlanıp izlenmezse, otomatik düzeltmeler istemeden verinin anlamını veya bağlamını değiştirebilir ve alt süreç sorunlarına yol açabilir.

Kendini onaran verinin temel risklerinden biri, veri düzeltme sürecine önyargı veya hataların dahil edilmesidir. Yapay zeka modelleri, diğer yazılım sistemleri gibi, eğitim verilerinde mevcut olan veya algoritmaların tasarımı yoluyla ortaya çıkan önyargılara tabi olabilir. Bu önyargılar tespit edilip azaltılmazsa, kendini onaran veri süreci boyunca yayılabilir ve çarpık veya yanlış veri değişikliklerine neden olabilir.

Örneğin, müşteri demografik verilerindeki tutarsızlıkları düzeltmekle görevlendirilmiş bir kendi kendini onaran veri işçisini ele alalım. Eğer yapay zeka modeli geçmiş verilerden önyargıları öğrendiyse, örneğin belirli meslekleri veya gelir seviyelerini belirli cinsiyetler veya etnik kökenlerle ilişkilendirme gibi, yanlış varsayımlarda bulunabilir ve verileri bu önyargıları pekiştirecek şekilde değiştirebilir. Bu durum, hatalı müşteri profillerine, yanlış yönlendirilmiş iş kararlarına ve potansiyel olarak ayrımcı sonuçlara yol açabilir.

Bir diğer potansiyel istenmeyen sonuç, veri düzeltme süreci sırasında değerli bilgilerin veya bağlamın kaybolmasıdır. Kendi kendini onaran veri teknikleri genellikle tutarlılığı sağlamak için verileri standartlaştırmaya ve normalleştirmeye odaklanır. Ancak bazı durumlarda, orijinal veriler tam resmi anlamak için önemli olan nüansları, istisnaları

veya bağlamsal bilgileri içerebilir. Körü körüne standardizasyonu uygulayan otomatik düzeltmeler, bu değerli bilgileri istemeden kaldırabilir veya belirsizleştirebilir.

Örneğin, tıbbi kayıtlardaki tutarsızlıkları düzeltmekle sorumlu bir kendi kendini onaran veri işçisini düşünün. İşçi, nadir görülen bir hastalığı olan veya olağandışı bir tedavi planı olan bir hastanın tıbbi geçmişiyle karşılaştığında, verileri daha yaygın bir kalıba uydurmak için normalleştirmeye çalışabilir. Ancak bunu yaparken, hastanın benzersiz durumunu doğru bir şekilde temsil etmek için kritik öneme sahip olan özel detayları ve bağlamı kaybedebilir. Bu bilgi kaybı, hasta bakımı ve tıbbi karar verme açısından ciddi sonuçlar doğurabilir.

İstenmeyen sonuçların risklerini azaltmak için, kendi kendini onaran veri tekniklerini tasarlarken ve uygularken proaktif bir yaklaşım benimsemek esastır:

1. **Kapsamlı Test ve Doğrulama**: Kendi kendini onaran veri işçilerini üretime almadan önce, davranışlarını çeşitli senaryolara karşı kapsamlı bir şekilde test etmek ve doğrulamak çok önemlidir. Bu, çeşitli uç durumları, istisnaları ve potansiyel önyargıları kapsayan temsili veri setleriyle test yapmayı içerir. Titiz testler, istenmeyen sonuçları gerçek dünya verilerini etkilemeden önce belirlemeye ve ele almaya yardımcı olur.
2. **Sürekli İzleme ve Değerlendirme**: İstenmeyen sonuçları zamanla tespit etmek ve azaltmak için sürekli izleme ve değerlendirme mekanizmalarının uygulanması esastır. Kendi kendini onaran veri süreçlerinin sonuçlarını düzenli olarak gözden geçirmek, alt sistemler ve karar verme üzerindeki etkiyi analiz etmek ve paydaşlardan geri bildirim toplamak, olumsuz etkileri belirlemeye ve zamanında düzeltici önlemler almaya yardımcı olabilir. Kuruluşunuzda operasyonel gösterge panelleri varsa, otomatik veri değişikliklerileriyle ilgili açıkça görülebilir metriklerin eklenmesi muhtemelen iyi bir fikirdir. Normal veri değişikliği aktivitesinden büyük sapmalara bağlı alarmlar eklemek muhtemelen daha da iyi bir fikirdir!
3. **İnsan Gözetimi ve Müdahalesi**: Kendi kendini onaran veri sürecinde insan gözetimini ve müdahale edebilme yeteneğini korumak çok önemlidir. Otomasyon

verimliliği büyük ölçüde artırabilirken, özellikle kritik veya hassas alanlarda yapay zeka modelleri tarafından yapılan düzeltmeleri insan uzmanların gözden geçirmesi ve doğrulaması önemlidir. İnsan muhakemesi ve alan uzmanlığı, ortaya çıkabilecek istenmeyen sonuçları belirlemeye ve ele almaya yardımcı olabilir.

4. **Açıklanabilir Yapay Zeka (XAI) ve Şeffaflık**: Önceki alt bölümde tartışıldığı gibi, açıklanabilir yapay zeka tekniklerini dahil etmek ve kendi kendini onaran veri sürecinde şeffaflığı sağlamak, istenmeyen sonuçları azaltmaya yardımcı olabilir. Veri düzeltmeleri için net açıklamalar sağlayarak ve kapsamlı denetim izleri tutarak, kuruluşlar yapay zeka modellerinin yaptığı değişikliklerin arkasındaki mantığı daha iyi anlayabilir ve izleyebilir.
5. **Aşamalı ve Yinelemeli Yaklaşım**: Kendi kendini onaran veriye aşamalı ve yinelemeli bir yaklaşım benimsemek, istenmeyen sonuçların riskini en aza indirmeye yardımcı olabilir. Otomatik düzeltmeleri bir kerede tüm veri setine uygulamak yerine, teknikler etkili ve güvenilir olduğunu kanıtladıkça veri alt kümesiyle başlayıp kapsamı kademeli olarak genişletin. Bu, süreç boyunca dikkatli izleme ve ayarlama yapılmasına olanak tanır ve istenmeyen sonuçların etkisini azaltır.
6. **İşbirliği ve Geri Bildirim**: Farklı alanlardan paydaşları sürece dahil etmek ve kendi kendini onaran veri süreci boyunca işbirliği ve geri bildirimi teşvik etmek, istenmeyen sonuçları belirlemeye ve ele almaya yardımcı olabilir. Alan uzmanlarından, veri tüketicilerinden ve son kullanıcılardan düzenli olarak girdi almak, veri düzeltmelerinin gerçek dünyadaki etkisi hakkında değerli bilgiler sağlayabilir ve gözden kaçmış olabilecek sorunları vurgulayabilir.

İstenmeyen sonuçların riskini proaktif olarak ele alarak ve uygun önlemleri uygulayarak, kuruluşlar potansiyel olumsuz etkileri en aza indirirken kendi kendini onaran veri tekniklerinin faydalarından yararlanabilir. Kendi kendini onaran veriye yinelemeli ve işbirlikçi bir süreç olarak yaklaşmak, teknikleri sürekli izlemek,

değerlendirmek ve istenen sonuçlarla uyumlu olmalarını ve verilerin bütünlüğünü ve güvenilirliğini korumalarını sağlamak için iyileştirmek önemlidir.

Kendi kendini onaran veri desenlerinin kullanımını düşünürken, bu faktörleri dikkatle değerlendirmek ve faydaları potansiyel riskler ve sınırlamalarla karşılaştırmak esastır. Bazı durumlarda, otomatik düzeltmeleri insan gözetimi ve müdahalesiyle birleştiren hibrit bir yaklaşım en uygun çözüm olabilir.

Ayrıca, kendi kendini onaran veri tekniklerinin sağlam veri doğrulama, girdi temizleme ve hata işleme mekanizmalarının yerini alması amaçlanmadığını belirtmek önemlidir. Bu temel uygulamalar veri bütünlüğü ve güvenliği için kritik öneme sahip olmaya devam etmektedir. Kendi kendini onaran veri, bu mevcut önlemleri destekleyebilecek ve geliştirebilecek tamamlayıcı bir yaklaşım olarak görülmelidir.

Sonuç olarak, kendi kendini onaran veri desenlerini kullanma kararı, uygulamanızın özel gereksinimleri, kısıtlamaları ve önceliklerine bağlıdır. Yukarıda belirtilen hususları dikkatle değerlendirerek ve bunları uygulamanızın hedefleri ve mimarisiyle uyumlu hale getirerek, kendi kendini onaran veri tekniklerini ne zaman ve nasıl etkili bir şekilde kullanacağınız konusunda bilinçli kararlar alabilirsiniz.

Bağlamsal İçerik Üretimi

Bağlamsal İçerik Üretimi kalıpları, uygulamalarda dinamik ve bağlama özel içerik üretmek için büyük dil modellerinin (BDM) gücünden yararlanır. Bu kalıp kategorisi, kullanıcılara özel ihtiyaçlarına, tercihlerine ve hatta uygulamayla önceki ve mevcut etkileşimlerine dayalı olarak kişiselleştirilmiş ve alakalı içerik sunmanın önemini kabul eder.

> Bu yaklaşımda "içerik" terimi, hem birincil içeriği (yani blog yazıları, makaleler vb.) hem de birincil içeriğe yönelik öneriler gibi meta-içeriği ifade eder.

Bağlamsal İçerik Üretimi kalıpları, kullanıcı etkileşim seviyelerinizi artırmada, kişiselleştirilmiş deneyimler sunmada ve hem sizin hem de kullanıcılarınız için içerik oluşturma görevlerini otomatikleştirmede çok önemli bir rol oynayabilir. Bu bölümde

açıkladığımız kalıpları kullanarak, bağlama ve girdilere gerçek zamanlı olarak uyum sağlayan, dinamik olarak içerik üreten uygulamalar oluşturabilirsiniz.

Kalıplar, BDM'leri kullanıcı arayüzünden (bazen "chrome" olarak adlandırılır) e-postalar ve diğer bildirim türlerine ve herhangi bir içerik üretim işlem hattına kadar uzanan uygulama çıktılarına entegre ederek çalışır.

Bir kullanıcı uygulamayla etkileşime girdiğinde veya belirli bir içerik talebi tetiklediğinde, uygulama kullanıcı tercihleri, önceki etkileşimler veya belirli komutlar gibi ilgili bağlamı yakalar. Bu bağlamsal bilgi daha sonra gerekli şablonlar veya yönergelerle birlikte BDM'ye beslenir ve aksi takdirde sabit kodlanması, bir veritabanında saklanması veya algoritmik olarak üretilmesi gereken metinsel çıktıyı üretmek için kullanılır.

BDM tarafından üretilen içerik, kişiselleştirilmiş öneriler, dinamik ürün açıklamaları, özelleştirilmiş e-posta yanıtları veya hatta tam makaleler ya da blog yazıları gibi çeşitli biçimler alabilir. Bu içeriğin en radikal kullanımlarından biri, bir yıldan fazla bir süre önce öncülük ettiğim form etiketleri, araç ipuçları ve diğer açıklayıcı metin gibi kullanıcı arayüzü öğelerinin dinamik olarak oluşturulmasıdır.

Kişiselleştirme

Bağlamsal İçerik Üretimi kalıplarının temel faydalarından biri, kullanıcılara son derece kişiselleştirilmiş deneyimler sunma yeteneğidir. Bu kalıplar, kullanıcıya özel bağlama dayalı içerik üreterek, uygulamaların içeriği her kullanıcının ilgi alanlarına, tercihlerine ve etkileşimlerine göre uyarlamasını sağlar.

Kişiselleştirme, sadece kullanıcının adını genel içeriğe eklemekten çok daha fazlasıdır. Her kullanıcı hakkında mevcut olan zengin bağlamı kullanarak, onların özel ihtiyaç ve istekleriyle örtüşen içerik üretmeyi içerir. Bu bağlam çeşitli faktörleri içerebilir, örneğin:

1. **Kullanıcı Profil Bilgileri:** Bu tekniğin en genel uygulama düzeyinde, demografik

veriler, ilgi alanları, tercihler ve diğer profil özellikleri, kullanıcının geçmişi ve özellikleriyle uyumlu içerik üretmek için kullanılabilir.

2. **Davranışsal Veriler**: Kullanıcının uygulama ile geçmiş etkileşimleri, görüntülenen sayfalar, tıklanan bağlantılar veya satın alınan ürünler gibi veriler, davranışları ve ilgi alanları hakkında değerli bilgiler sağlayabilir. Bu veriler, kullanıcının etkileşim kalıplarını yansıtan ve gelecekteki ihtiyaçlarını tahmin eden içerik önerileri üretmek için kullanılabilir.
3. **Bağlamsal Faktörler**: Kullanıcının mevcut konumu, cihazı, günün saati veya hatta hava durumu gibi mevcut bağlamı, içerik üretim sürecini etkileyebilir. Örneğin, bir seyahat uygulaması, kullanıcının mevcut konumuna ve hâkim hava koşullarına dayalı olarak kişiselleştirilmiş öneriler üretebilen bir yapay zeka çalışanına sahip olabilir.

Bu bağlamsal faktörlerden yararlanarak, Bağlamsal İçerik Üretimi kalıpları, uygulamaların her bir kullanıcı için özel olarak hazırlanmış gibi görünen içerik sunmasını sağlar. Bu düzeyde kişiselleştirmenin birkaç önemli faydası vardır:

1. **Artan Etkileşim**: Kişiselleştirilmiş içerik, kullanıcıların dikkatini çeker ve onları uygulama ile etkileşimde tutar. Kullanıcılar içeriğin alakalı olduğunu ve doğrudan kendi ihtiyaçlarına hitap ettiğini hissettiklerinde, uygulama ile daha fazla zaman geçirme ve özelliklerini keşfetme olasılıkları daha yüksektir.
2. **Gelişmiş Kullanıcı Memnuniyeti**: Kişiselleştirilmiş içerik, uygulamanın kullanıcının özel gereksinimlerini anladığını ve önemsediğini gösterir. Yardımcı, bilgilendirici ve ilgi alanlarıyla uyumlu içerik sunarak, uygulama kullanıcı memnuniyetini artırabilir ve kullanıcılarıyla daha güçlü bir bağ kurabilir.
3. **Daha Yüksek Dönüşüm Oranları**: E-ticaret veya pazarlama uygulamaları bağlamında, kişiselleştirilmiş içerik dönüşüm oranlarını önemli ölçüde etkileyebilir. Kullanıcılara tercihlerine ve davranışlarına göre uyarlanmış ürünler, teklifler veya öneriler sunarak, uygulama kullanıcıların satın alma veya bir hizmete kaydolma gibi istenen eylemleri gerçekleştirme olasılığını artırabilir.

Üretkenlik

Bağlamsal İçerik Üretimi kalıpları, yaratıcı süreçlerde manuel içerik üretimi ve düzenleme ihtiyacını azaltarak belirli türdeki üretkenliği önemli ölçüde artırabilir. BDM'lerin gücünden yararlanarak, içerik oluşturucularınızın ve geliştiricilerinizin sıkıcı manuel işlere harcamak zorunda kalacağı zaman ve çabadan tasarruf ederek, ölçeklenebilir yüksek kaliteli içerik üretebilirsiniz.

Geleneksel olarak, içerik üreticilerinin uygulamanın gereksinimlerini ve kullanıcı beklentilerini karşılamak için içeriği araştırması, yazması, düzenlemesi ve biçimlendirmesi gerekir. Bu süreç, özellikle içerik hacmi büyüdükçe zaman alıcı ve kaynak yoğun olabilir.

Ancak, Bağlamsal İçerik Üretimi kalıplarıyla içerik oluşturma süreci büyük ölçüde otomatikleştirilebilir. Büyük Dil Modelleri, verilen komutlar ve yönergeler doğrultusunda tutarlı, dilbilgisi açısından doğru ve bağlamsal olarak ilgili içerik üretebilir. Bu otomasyon birkaç üretkenlik avantajı sunar:

1. **Azaltılmış Manuel Çaba**: İçerik üretimi görevlerini Büyük Dil Modellerine devrederek, içerik üreticileri içerik stratejisi, fikir üretimi ve kalite güvencesi gibi daha üst düzey görevlere odaklanabilir. Büyük Dil Modeline gerekli bağlamı, şablonları ve yönergeleri sağlayabilir ve asıl içerik üretimini ona bırakabilirler. Bu, yazma ve düzenleme için gereken manuel çabayı azaltarak içerik üreticilerinin daha üretken ve verimli olmasını sağlar.
2. **Daha Hızlı İçerik Üretimi**: Büyük Dil Modelleri, insan yazarlardan çok daha hızlı içerik üretebilir. Doğru komutlar ve yönergelerle bir Büyük Dil Modeli, saniyeler veya dakikalar içinde birden fazla içerik parçası üretebilir. Bu hız, uygulamaların çok daha hızlı bir tempoda içerik üretmesini sağlayarak kullanıcıların taleplerini ve sürekli değişen dijital ortamı yakalamasına olanak tanır.

> Daha hızlı içerik üretimi, internetin kimsenin okumadığı içeriklerle dolup taşması şeklinde bir "ortak malların trajedisi" durumuna mı yol açıyor? Ne yazık ki, cevabın evet olduğundan şüpheleniyorum.

3. **Tutarlılık ve Kalite**: Büyük Dil Modelleri, içeriği stil, ton ve kalite açısından tutarlı olacak şekilde kolaylıkla düzenleyebilir. Net yönergeler ve örnekler sağlandığında, belirli türdeki uygulamalar (örneğin haber odası, PR vb.) insan tarafından üretilen içeriklerinin marka sesiyle uyumlu olmasını ve istenen kalite standartlarını karşılamasını sağlayabilir. Bu tutarlılık, kapsamlı düzenleme ve revizyonlara olan ihtiyacı azaltarak içerik oluşturma sürecinde zaman ve çabadan tasarruf sağlar.
4. **İterasyon ve Optimizasyon**: Bağlamsal İçerik Üretimi kalıpları hızlı iterasyon ve içerik optimizasyonuna olanak tanır. Büyük Dil Modeline sağlanan komutları, şablonları veya yönergeleri ayarlayarak, uygulamalarınız geçmişte hiç mümkün olmayan bir şekilde otomatik olarak içerik varyasyonları üretebilir ve farklı yaklaşımları test edebilir. Bu tekrarlayıcı süreç, içerik stratejilerinin daha hızlı denenmesine ve iyileştirilmesine olanak tanır ve zamanla daha etkili ve ilgi çekici içerik üretilmesini sağlar. Bu özel teknik, hemen çıkma oranları ve etkileşime göre yaşayan ve ölen e-ticaret gibi uygulamalar için tam bir oyun değiştirici olabilir

Bağlamsal İçerik Üretimi kalıplarının üretkenliği büyük ölçüde artırabileceğini, ancak insan katılımı ihtiyacını tamamen ortadan kaldırmadığını belirtmek önemlidir. İçerik üreticileri ve editörler, genel içerik stratejisini tanımlamada, Büyük Dil Modeline rehberlik sağlamada ve üretilen içeriğin kalitesini ve uygunluğunu sağlamada hala çok önemli bir rol oynamaktadır.

İçerik oluşturmanın daha tekrarlayıcı ve zaman alıcı yönlerini otomatikleştirerek,

Bağlamsal İçerik Üretimi kalıpları daha yüksek değerli görevlere yönlendirilebilecek değerli insan zamanını ve kaynaklarını serbest bırakır. Bu artan üretkenlik, içerik oluşturma iş akışlarını optimize ederken kullanıcılara daha kişiselleştirilmiş ve ilgi çekici içerik sunmanızı sağlar.

Hızlı İterasyon ve Deney

Bağlamsal İçerik Üretimi kalıpları, farklı içerik varyasyonlarıyla hızlı bir şekilde iterasyon yapmanıza ve deney yapmanıza olanak tanıyarak, içerik stratejinizin daha hızlı optimize edilmesini ve iyileştirilmesini sağlar. Modele sağlanan bağlamı, şablonları veya yönergeleri ayarlayarak saniyeler içinde birden fazla içerik versiyonu üretebilirsiniz.

Bu hızlı iterasyon yeteneği birkaç önemli avantaj sunar:

1. **Test Etme ve Optimizasyon**: İçerik varyasyonlarını hızlı bir şekilde üretme yeteneği ile farklı yaklaşımları kolayca test edebilir ve etkinliklerini ölçebilirsiniz. Örneğin, belirli bir kullanıcı segmentine veya bağlama göre uyarlanmış bir ürün açıklamasının veya pazarlama mesajının birden fazla versiyonunu üretebilirsiniz. Tıklama oranları veya dönüşüm oranları gibi kullanıcı etkileşim metriklerini analiz ederek, en etkili içerik varyasyonlarını belirleyebilir ve içerik stratejinizi buna göre optimize edebilirsiniz.

2. **A/B Testi:** Bağlamsal İçerik Üretimi kalıpları, içeriğin sorunsuz A/B testine olanak tanır. İki veya daha fazla içerik varyasyonu üretebilir ve bunları farklı kullanıcı gruplarına rastgele sunabilirsiniz. Her varyasyonun performansını karşılaştırarak, hedef kitlenize en iyi uyan içeriği belirleyebilirsiniz. Bu veri odaklı yaklaşım, kullanıcı etkileşimini en üst düzeye çıkarmak ve istenen sonuçlara ulaşmak için bilinçli kararlar almanızı ve içeriğinizi sürekli olarak iyileştirmenizi sağlar.

3. **Kişiselleştirme Deneyleri**: Hızlı iterasyon ve deney, özellikle kişiselleştirme söz konusu olduğunda çok değerlidir. Bağlamsal İçerik Üretimi kalıplarıyla, farklı kullanıcı segmentleri, tercihleri veya davranışlarına dayalı olarak hızlı bir şekilde kişiselleştirilmiş içerik varyasyonları üretebilirsiniz. Farklı kişiselleştirme stratejileriyle deney yaparak, bireysel kullanıcıları çekmek ve özelleştirilmiş deneyimler sunmak için en etkili yaklaşımları belirleyebilirsiniz.
4. **Değişen Trendlere Uyum Sağlama**: Hızlı iterasyon ve deney yapabilme yeteneği, değişen trendlere ve kullanıcı tercihlerine çevik bir şekilde uyum sağlamanızı mümkün kılar. Yeni konular, anahtar kelimeler veya kullanıcı davranışları ortaya çıktıkça, bu trendlerle uyumlu içerik hızlıca üretebilirsiniz. İçeriğinizi sürekli olarak test edip geliştirerek, sürekli gelişen dijital ortamda güncel kalabilir ve rekabet avantajınızı koruyabilirsiniz.
5. **Maliyet Etkin Deneyler**: Geleneksel içerik deneyleri, içerik üreticilerinin farklı varyasyonları manuel olarak geliştirmesi ve test etmesi gerektiğinden, genellikle önemli zaman ve kaynak gerektirir. Ancak, Bağlamsal İçerik Üretimi desenleriyle, deney maliyeti büyük ölçüde azalır. LLM'ler hızlı ve ölçekli bir şekilde içerik varyasyonları üreterek, önemli maliyetler olmadan çok çeşitli fikir ve yaklaşımları keşfetmenizi sağlar.

Hızlı iterasyon ve deneylerden en iyi şekilde yararlanmak için, iyi tanımlanmış bir deney çerçevesinin olması önemlidir. Bu çerçeve şunları içermelidir:

- Her deney için net hedefler ve hipotezler
- İçerik performansını ölçmek için uygun metrikler ve takip mekanizmaları
- İlgili içerik varyasyonlarının doğru kullanıcılara sunulmasını sağlayan segmentasyon ve hedefleme stratejileri
- Deneysel verilerden içgörü elde etmek için analiz ve raporlama araçları
- Öğrenilenleri ve optimizasyonları içerik stratejinize dahil etme süreci

Hızlı iterasyon ve deneyleri benimseyerek, içeriğinizi sürekli olarak iyileştirebilir ve optimize edebilir, böylece uygulamanızın hedeflerine ulaşmada etkili, ilgi çekici ve

güncel kalmasını sağlayabilirsiniz. İçerik oluşturmaya yönelik bu çevik yaklaşım, trendin önünde kalmanızı ve olağanüstü kullanıcı deneyimleri sunmanızı sağlar.

Ölçeklenebilirlik ve Verimlilik

Uygulamalar büyüdükçe ve kişiselleştirilmiş içeriğe olan talep arttıkça, bağlamsal içerik üretimi desenleri, içerik oluşturmanın verimli bir şekilde ölçeklenmesini sağlar. LLM'ler, insan kaynaklarında orantılı bir artış gerektirmeden, çok sayıda kullanıcı ve bağlam için eş zamanlı olarak içerik üretebilir. Bu ölçeklenebilirlik, uygulamaların içerik oluşturma kapasitelerini zorlamadan, büyüyen bir kullanıcı tabanına kişiselleştirilmiş deneyimler sunmasına olanak tanır.

Bağlamsal içerik üretiminin uygulamanızı "anında" uluslararasılaştırmak için etkili bir şekilde kullanılabileceğini unutmayın. Aslında, bir yıldan kısa bir süre içinde Instant18n Gem'imi kullanarak Olympia'yı yarım düzineden fazla dilde sunmak için tam olarak bunu yaptım.

Yapay Zeka Destekli Yerelleştirme

Bir anlığına övünmeme izin verirseniz, Rails uygulamaları için geliştirdiğim Instant18n kütüphanesinin, uygulama geliştirmede yapay zekanın dönüştürücü potansiyelini gösteren "Bağlamsal İçerik Üretimi" deseninin çığır açan bir örneği olduğunu düşünüyorum. Bu gem, Rails uygulamalarında uluslararasılaştırma ve yerelleştirmenin nasıl ele alındığını devrimleştirmek için OpenAI'nin GPT büyük dil modelinin gücünden yararlanıyor.

Geleneksel olarak, bir Rails uygulamasını uluslararasılaştırmak, çeviri anahtarlarını manuel olarak tanımlamayı ve desteklenen her dil için karşılık gelen çevirileri sağlamayı içerir. Bu süreç zaman alıcı, kaynak yoğun ve tutarsızlıklara açık olabilir. Ancak Instant18n gem'i ile yerelleştirme paradigması tamamen yeniden tanımlanıyor.

Büyük bir dil modelini entegre ederek, Instant18n gem'i metnin bağlamına ve anlamına dayalı olarak anında çeviriler üretmenizi sağlar. Önceden tanımlanmış çeviri anahtarlarına ve statik çevirilere güvenmek yerine, gem metni yapay zeka gücünü kullanarak dinamik olarak çevirir. Bu yaklaşım birkaç önemli fayda sunar:

1. **Sorunsuz Yerelleştirme**: Instant18n gem'i ile geliştiricilerin desteklenen her dil için çeviri dosyalarını manuel olarak tanımlayıp bakımını yapmasına gerek kalmaz. Gem, sağlanan metin ve hedef dile göre otomatik olarak çeviriler oluşturarak, yerelleştirme sürecini zahmetsiz ve sorunsuz hale getirir.
2. **Bağlamsal Doğruluk**: Yapay zekaya çevrilen metnin nüanslarını anlaması için yeterli bağlam verilebilir. Çevreleyen bağlamı, deyimleri ve kültürel referansları dikkate alarak doğru, doğal ve bağlama uygun çeviriler üretebilir.
3. **Kapsamlı Dil Desteği**: Instant18n gem'i, GPT'nin geniş bilgi birikimi ve dilsel yeteneklerinden yararlanarak, çok sayıda dile çeviri yapılmasını sağlar. İspanyolca ve Fransızca gibi yaygın dillerden, Klingon ve Elf dili gibi daha az bilinen veya kurgusal dillere kadar, gem çok çeşitli çeviri gereksinimlerini karşılayabilir.
4. **Esneklik ve Yaratıcılık**: Gem, geleneksel dil çevirilerinin ötesine geçerek yaratıcı ve alışılmadık yerelleştirme seçeneklerine olanak tanır. Geliştiriciler metni çeşitli stillere, lehçelere veya hatta kurgusal dillere çevirebilir, bu da benzersiz kullanıcı deneyimleri ve ilgi çekici içerik için yeni olanaklar sunar.
5. **Performans Optimizasyonu**: Instant18n gem'i, performansı artırmak ve tekrarlanan çevirilerin yükünü azaltmak için önbellekleme mekanizmaları içerir. Çevrilen metin önbelleğe alınır, böylece aynı çeviri için sonraki istekler gereksiz API çağrıları olmadan hızlı bir şekilde sunulabilir.

Instant18n gem'i, yapay zekayı kullanarak dinamik olarak yerelleştirilmiş içerik üreterek "Bağlamsal İçerik Üretimi" deseninin gücünü örneklemektedir. Yapay zekanın bir Rails uygulamasının temel işlevselliğine nasıl entegre edilebileceğini ve geliştiricilerin uluslararasılaştırma ve yerelleştirmeye yaklaşımını nasıl dönüştürdüğünü göstermektedir.

Manuel çeviri yönetimi ihtiyacını ortadan kaldırarak ve bağlama dayalı anında çeviriler sağlayarak, Instant18n gem'i geliştiricilere önemli ölçüde zaman ve çaba tasarrufu sağlar. Geliştiricilerin, yerelleştirme yönünün sorunsuz ve doğru bir şekilde ele alınmasını sağlarken uygulamalarının temel özelliklerini geliştirmeye odaklanmalarına olanak tanır.

Kullanıcı Testi ve Geri Bildirimin Önemi

Son olarak, kullanıcı testi ve geri bildirimin önemini her zaman aklınızda tutun. Bağlamsal içerik oluşturmanın kullanıcı beklentilerini karşıladığını ve uygulamanın hedefleriyle uyumlu olduğunu doğrulamak çok önemlidir. Kullanıcı içgörüleri ve analitiğe dayalı olarak oluşturulan içeriği sürekli geliştirin ve iyileştirin. Siz ve ekibiniz tarafından manuel olarak doğrulanması imkansız olacak büyük ölçekte dinamik içerik oluşturuyorsanız, kullanıcıların tuhaf veya yanlış olan içeriği, nedeniyle birlikte bildirebilecekleri geri bildirim mekanizmaları eklemeyi düşünün. Bu değerli geri bildirimler, içeriği oluşturan bileşende düzeltmeler yapmakla görevlendirilmiş bir yapay zeka işçisine bile beslenebilir!

Üretici Kullanıcı Arayüzü

Dikkat günümüzde o kadar değerli ki, etkili kullanıcı etkileşimi artık yalnızca sorunsuz ve sezgisel değil, aynı zamanda bireysel ihtiyaçlara, tercihlere ve bağlamlara göre yüksek düzeyde kişiselleştirilmiş yazılım deneyimleri gerektiriyor. Sonuç olarak, tasarımcılar ve geliştiriciler giderek artan bir şekilde, her kullanıcının benzersiz gereksinimlerine *ölçekli bir şekilde* uyum sağlayabilen ve hitap edebilen kullanıcı arayüzleri oluşturma zorluğuyla karşı karşıya kalıyorlar.

Üretici Kullanıcı Arayüzü (GenUI), kullanıcı arayüzü tasarımında gerçekten devrim niteliğinde bir yaklaşımdır ve anında yüksek düzeyde kişiselleştirilmiş ve dinamik kullanıcı deneyimleri oluşturmak için büyük dil modellerinin (LLM) gücünden yararlanır. Bu kitapta GenUI hakkında en azından bir giriş yapmak istedim, çünkü bunun uygulama tasarımı ve çerçeveleri alanında şu anda var olan en bakir fırsatlardan biri olduğuna inanıyorum. Bu özel niş alanda düzinelerce veya daha fazla yeni başarılı ticari ve açık kaynaklı projenin ortaya çıkacağına ikna olmuş durumdayım.

Özünde, GenUI, Bağlamsal İçerik Üretimi ilkelerini gelişmiş yapay zeka teknikleriyle birleştirerek, kullanıcının bağlamını, tercihlerini ve hedeflerini derinlemesine anlayarak metin, görsel ve düzenler gibi kullanıcı arayüzü öğelerini dinamik olarak üretir. GenUI, tasarımcıların ve geliştiricilerin, kullanıcı etkileşimlerine yanıt olarak uyum sağlayan ve gelişen arayüzler oluşturmasına olanak tanıyarak, daha önce ulaşılamaz olan bir kişiselleştirme düzeyi sağlar.

GenUI, kullanıcı arayüzü tasarımına yaklaşımımızda temel bir değişimi temsil eder. Kitleler için tasarım yapmak yerine, GenUI bireye özel tasarım yapmamıza olanak tanır. Kişiselleştirilmiş içerik ve arayüzler, her kullanıcıyla daha derin düzeyde rezonans oluşturan, etkileşimi, memnuniyeti ve sadakati artıran kullanıcı deneyimleri yaratma potansiyeline sahiptir.

En yeni teknolojilerden biri olarak, GenUI'ye geçiş, kavramsal ve pratik zorluklarla doludur. Yapay zekayı tasarım sürecine entegre etmek, üretilen arayüzlerin sadece kişiselleştirilmiş değil, aynı zamanda kullanılabilir, erişilebilir ve genel marka ve kullanıcı deneyimiyle uyumlu olmasını sağlamak, tüm bunlar GenUI'yi çoğunluğun değil, azınlığın uğraşı haline getiren zorluklardır. Ayrıca, yapay zekanın dahil olması, veri gizliliği, şeffaflık ve hatta etik sonuçlar hakkında sorular ortaya çıkarır.

Zorluklara rağmen, ölçekli kişiselleştirilmiş deneyimler, dijital ürün ve hizmetlerle etkileşim kurma şeklimizi tamamen dönüştürme gücüne sahiptir. Bu, yetenekleri, geçmişleri veya tercihleri ne olursa olsun, kullanıcıların çeşitli ihtiyaçlarına hitap eden kapsayıcı ve erişilebilir arayüzler oluşturma olanaklarını açar.

Bu bölümde, GenUI kavramını inceleyerek bazı belirleyici özelliklerini, temel faydalarını ve potansiyel zorluklarını ele alacağız. GenUI'nin en temel ve erişilebilir formuyla başlıyoruz: geleneksel olarak tasarlanmış ve uygulanmış kullanıcı arayüzleri için metin içeriği üretmek.

Kullanıcı Arayüzleri İçin Metin Üretimi

Uygulamanızın arayüz öğelerinde bulunan form etiketleri, ipucu metinleri ve açıklayıcı metinler gibi metin öğeleri, genellikle şablonlara veya UI bileşenlerine sabit olarak kodlanır ve tüm kullanıcılar için tutarlı ancak genel bir deneyim sunar. Bağlamsal içerik üretimi kalıplarını kullanarak, bu statik öğeleri dinamik, bağlama duyarlı ve kişiselleştirilmiş bileşenlere dönüştürebilirsiniz.

Kişiselleştirilmiş Formlar

Formlar, web ve mobil uygulamaların yaygın bir parçasıdır ve kullanıcı girdisi toplamanın birincil yoludur. Ancak, geleneksel formlar genellikle kullanıcının özel bağlamı veya ihtiyaçlarıyla her zaman uyumlu olmayan standart etiketler ve alanlarla genel ve kişisel olmayan bir deneyim sunar. Kullanıcılar, ihtiyaç ve tercihlerine uyarlanmış hissettikleri formları doldurma olasılıkları daha yüksektir, bu da daha yüksek dönüşüm oranları ve kullanıcı memnuniyeti sağlar.

Bununla birlikte, kişiselleştirme ve tutarlılık arasında bir denge kurmak önemlidir. Formları bireysel kullanıcılara uyarlamak faydalı olabilirken, bir tanıdıklık ve öngörülebilirlik düzeyini korumak çok önemlidir. Kullanıcılar, kişiselleştirilmiş öğelerle bile formları kolayca tanıyabilmeli ve gezinebilmelidir.

İşte ilham verici bazı kişiselleştirilmiş form fikirleri:

Bağlamsal Alan Önerileri

GenUI, kullanıcının önceki etkileşimlerini, tercihlerini ve verilerini analiz ederek tahminler olarak akıllı alan önerileri sunabilir. Örneğin, kullanıcı daha önce teslimat adresini girdiyse, form ilgili alanları kaydedilmiş bilgileriyle otomatik olarak doldurabilir. Bu sadece zamandan tasarruf sağlamakla kalmaz, aynı zamanda uygulamanın kullanıcının tercihlerini anladığını ve hatırladığını gösterir.

Bir dakika, bu teknik yapay zeka kullanmadan da yapılabilecek bir şey değil mi? Elbette öyle, ancak bu tür bir işlevselliği yapay zeka ile yönlendirmenin iki önemli güzelliği var: 1) uygulamanın ne kadar kolay olabileceği ve 2) kullanıcı arayüzünüz değişip geliştikçe ne kadar dayanıklı kalabileceği.

Hadi teorik sipariş işleme sistemimiz için, kullanıcının doğru teslimat adresini proaktif olarak doldurmaya çalışan bir servis oluşturalım.

```ruby
class OrderShippingAddressSubscriber
  include Raix::ChatCompletion

  attr_accessor :order

  delegate :customer, to: :order

  DIRECTIVE = "You are a smart order processing assistant. Given the
  customer's order history, guess the most likely shipping address
  for the current order."

  def order_created(order)
    return unless order.pending? && order.shipping_address.blank?

    self.order = order

    transcript.clear
    transcript << { system: DIRECTIVE }
    transcript << { user: "Order History: #{order_history.to_json}" }
    transcript << { user: "Current Order: #{order.to_json}" }

    response = chat_completion
    apply_predicted_shipping_address(order, response)
  end

  private

  def apply_predicted_shipping_address(order, response)
    # extract the shipping address from the response...
    # ...and assume there's some sort of live update of the address fields
    order.update(shipping_address:)
  end
```

```
33
34    def order_history
35      customer.orders.successful.limit(100).map do |order|
36        {
37          date: order.date,
38          description: order.description,
39          shipping_address: order.shipping_address
40        }
41      end
42    end
43  end
```

Bu örnek oldukça basitleştirilmiş olsa da çoğu durum için işe yarayacaktır. Buradaki fikir, yapay zekanın tıpkı bir insanın yapacağı gibi tahmin yürütmesine izin vermektir. Ne demek istediğimi açıklığa kavuşturmak için, bazı örnek verilere bakalım:

```
1   Order History:
2   [
3     {"date": "2024-01-03", "description": "garden soil mix",
4      "shipping_address": "123 Country Lane, Rural Town"},
5     {"date": "2024-01-15", "description": "hardcover fiction novels",
6      "shipping_address": "456 City Apt, Metroville"},
7     {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
8      "789 Suburb St, Quietville"},
9     {"date": "2024-02-01", "description": "organic vegetables",
10     "shipping_address": "123 Country Lane, Rural Town"},
11    {"date": "2024-02-17", "description": "mystery thriller book set",
12     "shipping_address": "456 City Apt, Metroville"},
13    {"date": "2024-02-25", "description": "baby wipes",
14     "shipping_address": "789 Suburb St, Quietville"},
15    {"date": "2024-03-05", "description": "flower seeds",
16     "shipping_address": "123 Country Lane, Rural Town"},
17    {"date": "2024-03-20", "description": "biographies",
18     "shipping_address": "456 City Apt, Metroville"},
19    {"date": "2024-03-30", "description": "baby formula",
20     "shipping_address": "789 Suburb St, Quietville"},
21    {"date": "2024-04-12", "description": "lawn fertilizer",
22     "shipping_address": "123 Country Lane, Rural Town"},
23    {"date": "2024-04-22", "description": "science fiction novels",
24     "shipping_address": "456 City Apt, Metroville"},
```

```
  {"date": "2024-05-02", "description": "infant toys",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-05-14", "description": "outdoor grill",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-05-29", "description": "literary classics",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-06-11", "description": "baby clothes",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-07-01", "description": "watering can",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-07-18", "description": "non-fiction essays",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-07-28", "description": "baby bath items",
  "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-08-09", "description": "herb garden kit",
  "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-08-24", "description": "children's books",
  "shipping_address": "456 City Apt, Metroville"}
]
```

Verilerdeki örüntüyü fark ettiniz mi? Bu tür şeyler bir BDM için çocuk oyuncağı, garanti ederim. Bunu göstermek için, GPT-4'e bir "termometre" için en olası teslimat adresinin ne olduğunu soralım.

```
From the order history you've provided, it looks like the purchases are
generally clustered into three main types based on the shipping addresses:

123 Country Lane, Rural Town - This address often orders garden and
outdoor-related items like soil mix, vegetables, flower seeds, lawn
fertilizer, an outdoor grill, a watering can, and an herb garden kit.

456 City Apt, Metroville - This address tends to order various types of
books, indicating an interest in reading across genres like fiction,
thrillers, biographies, science fiction, literary classics, non-fiction
essays, and children's books.

789 Suburb St, Quietville - The purchases for this address are consistently
baby-related products, such as diapers, wipes, baby formula, infant toys,
baby clothes, and baby bath items.

```

```
Given these patterns, a digital thermometer could potentially be a fit for
any of the addresses based on different needs: outdoor temperature tracking,
a tool for a passionate reader interested in science gadgets, or a health
tool for a baby. However, considering its common use in monitoring health,
especially for babies, it's most likely to be purchased by someone who's
already buying baby care products.

Therefore, the most likely shipping address for a digital thermometer, based
on the purchasing patterns, would be 789 Suburb St, Quietville.
```

Bu iş için GPT-4 gibi pahalı bir modeli kullanmanın fazla abartılı olduğunu düşünüyorsanız, haklısınız! Aynı komutu Mistral 7B Instruct üzerinde denediğimde, saniyede 75 token hızında ve sadece 0,000218 USD gibi cüzi bir maliyetle aynı yanıtı üretti.

```
Based on the given order history, it appears that the customer has received
orders for baby-related items (diapers, wipes, formula, clothes, bath items)
at the address "789 Suburb St, Quietville" and garden-related items (soil,
seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
Town". The customer has also received books (fiction, non-fiction, literary
classics, mystery-thriller, science fiction, biographies, children's books)
at the address "456 City Apt, Metroville".

Given that the new order is for a "thermometer", which doesn't seem to fit
into any of the categories that the customer has previously ordered, it's
hard to accurately predict the shipping address based solely on the order
history. However, given the 50%-50% split between baby-related and
garden-related items, it could somewhat lean towards the Baby-related items
address ("789 Suburb St, Quietville"). But remember, this is an assumption
and cannot be definitively confirmed without more context or information.
```

Bu tekniğin getirdiği ek yük ve maliyet, ödeme sürecini daha sihirli hale getirmek için değer mi? Birçok çevrimiçi perakendeci için kesinlikle değer. Görünen o ki, özellikle taban fiyat rekabetine giren açık kaynak model barındırma sağlayıcıları sayesinde, yapay zeka hesaplama maliyetleri yalnızca düşmeye devam edecek.

Bu tür sohbet tamamlamalarını optimize etmek için Prompt Template ve StructuredIO ile birlikte Response Fencing kullanın.

Uyarlanabilir Alan Sıralaması

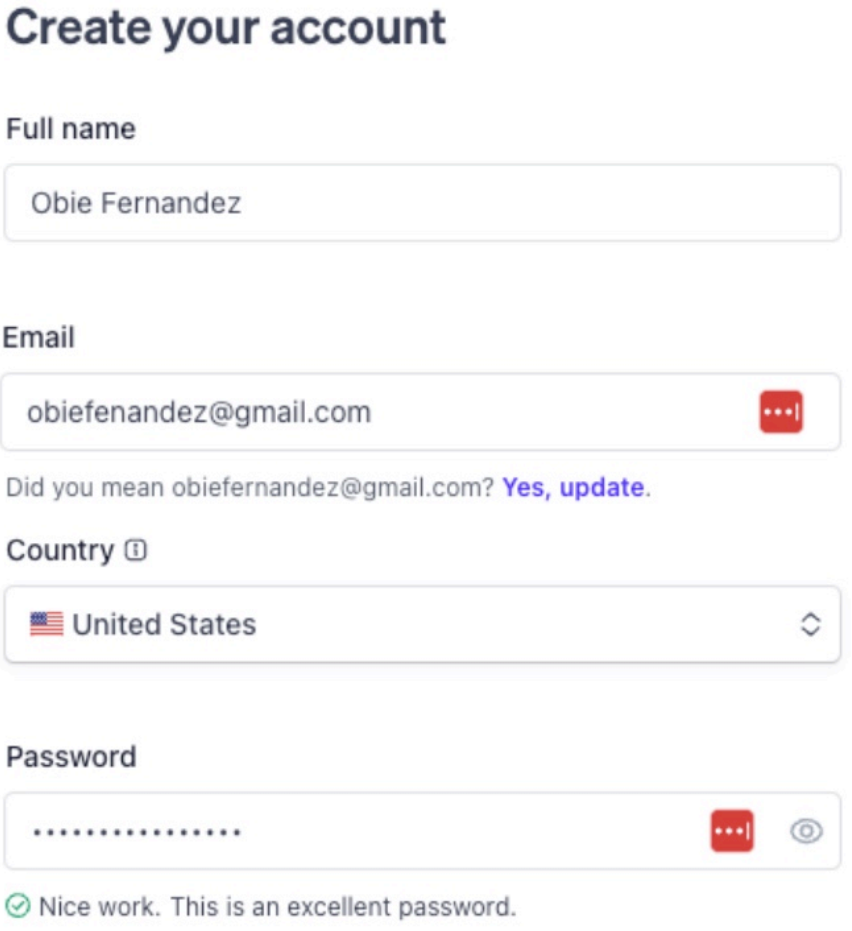
Form alanlarının sunulma sırası, kullanıcı deneyimini ve tamamlanma oranlarını önemli ölçüde etkileyebilir. GenUI ile kullanıcının bağlamına ve her alanın önemine göre alan sıralamasını dinamik olarak ayarlayabilirsiniz. Örneğin, bir kullanıcı fitness uygulaması için kayıt formunu dolduruyorsa, form fitness hedefleri ve tercihlerine ilişkin alanları önceliklendirebilir ve böylece süreci daha alakalı ve ilgi çekici hale getirebilir.

Kişiselleştirilmiş Mikrokopya

Formlardaki talimat metinleri, hata mesajları ve diğer mikrokopyalar da GenUI kullanılarak kişiselleştirilebilir. "Geçersiz e-posta adresi" gibi genel hata mesajları yerine, "Sipariş onayınızı almak için lütfen geçerli bir e-posta adresi girin" gibi daha yardımcı ve bağlamsal mesajlar oluşturabilirsiniz. Bu kişiselleştirilmiş dokunuşlar, form deneyimini daha kullanıcı dostu ve daha az sinir bozucu hale getirebilir.

Kişiselleştirilmiş Doğrulama

Kişiselleştirilmiş Mikrokopya ile aynı doğrultuda, sihirli görünen şekillerde formu doğrulamak için yapay zeka kullanabilirsiniz. Bir kullanıcı profil formundaki potansiyel hataları *anlamsal* düzeyde arayan bir yapay zeka düşünün.



Şekil 9. Gerçekleşen anlamsal doğrulamayı görebiliyor musunuz?

Aşamalı Gösterim

GenUI, kullanıcının bağlamına göre hangi form alanlarının gerekli olduğunu akıllıca belirleyebilir ve gerektiğinde ek alanları kademeli olarak gösterebilir. Bu aşamalı gösterim tekniği, bilişsel yükü azaltmaya ve form doldurma sürecini daha yönetilebilir hale getirmeye yardımcı olur. Örneğin, bir kullanıcı temel bir abonelik

için kaydoluyorsa, form başlangıçta yalnızca temel alanları gösterebilir ve kullanıcı ilerledikçe veya belirli seçenekleri seçtikçe ilgili ek alanlar dinamik olarak eklenebilir.

Bağlama Duyarlı Açıklayıcı Metin

İpucu baloncukları genellikle kullanıcılar belirli öğelerin üzerine geldiğinde veya etkileşime girdiğinde ek bilgi veya rehberlik sağlamak için kullanılır. "Bağlamsal İçerik Oluşturma" yaklaşımıyla, kullanıcının bağlamına uyum sağlayan ve ilgili bilgileri sunan ipucu baloncukları oluşturabilirsiniz. Örneğin, bir kullanıcı karmaşık bir özelliği keşfediyorsa, ipucu baloncuğu önceki etkileşimlerine veya beceri düzeyine göre kişiselleştirilmiş ipuçları veya örnekler sunabilir.

Talimatlar, açıklamalar veya yardım mesajları gibi açıklayıcı metinler, kullanıcının bağlamına göre dinamik olarak oluşturulabilir. Genel açıklamalar sunmak yerine, kullanıcının özel ihtiyaçlarına veya sorularına göre uyarlanmış metin oluşturmak için LLM'leri kullanabilirsiniz. Örneğin, bir kullanıcı bir sürecin belirli bir adımında zorlanıyorsa, açıklayıcı metin kişiselleştirilmiş rehberlik veya sorun giderme ipuçları sağlayabilir.

Mikrokopya, düğme etiketleri, hata mesajları veya onay istemler gibi kullanıcıları uygulamanızda yönlendiren küçük metin parçalarını ifade eder. Bağlamsal İçerik Oluşturma yaklaşımını mikrokopyaya uygulayarak, kullanıcının eylemlerine yanıt veren ve ilgili ve yardımcı metin sağlayan uyarlanabilir bir UI oluşturabilirsiniz. Örneğin, bir kullanıcı kritik bir eylem gerçekleştirmek üzereyse, onay istemi net ve kişiselleştirilmiş bir mesaj sağlamak için dinamik olarak oluşturulabilir.

Kişiselleştirilmiş açıklayıcı metin ve ipucu baloncukları, yeni kullanıcılar için başlangıç sürecini büyük ölçüde iyileştirebilir. Bağlama özel rehberlik ve örnekler sağlayarak, kullanıcıların uygulamayı hızlıca anlamasına ve gezinmesine yardımcı olabilir, öğrenme süresini kısaltabilir ve benimsemeyi artırabilirsiniz.

Dinamik ve bağlama duyarlı arayüz öğeleri de uygulamanın daha sezgisel ve ilgi çekici hissetmesini sağlayabilir. Eşlik eden metin kendi özel ihtiyaç ve ilgi alanlarına göre

uyarlandığında, kullanıcıların özelliklerle etkileşime girme ve bunları keşfetme olasılığı daha yüksektir.

Şimdiye kadar yapay zeka ile mevcut kullanıcı arayüzü paradigmalarını geliştirme fikirlerini ele aldık, peki ya kullanıcı arayüzlerinin nasıl tasarlandığını ve uygulandığını daha radikal bir şekilde yeniden düşünmek hakkında ne dersiniz?

Üretken Kullanıcı Arayüzünün Tanımlanması

Tasarımcıların sabit, statik arayüzler oluşturduğu geleneksel kullanıcı arayüzü tasarımının aksine, GenUI yazılımlarımızın gerçek zamanlı olarak gelişebilen ve uyum sağlayabilen esnek, kişiselleştirilmiş deneyimler sunduğu bir geleceğe işaret ediyor. Yapay zeka destekli bir konuşma arayüzünü her kullandığımızda, yapay zekanın kullanıcının özel ihtiyaçlarına uyum sağlamasına izin veriyoruz. GenUI, bu uyarlanabilirlik seviyesini yazılımın *görsel* arayüzüne uygulayarak işleri bir adım öteye taşıyor.

Günümüzde GenUI fikirleriyle çalışmanın mümkün olmasının nedeni, büyük dil modellerinin halihazırda programlamayı anlaması ve temel bilgilerinin kullanıcı arayüzü teknolojilerini ve çerçevelerini içermesidir. Asıl soru, büyük dil modellerinin her kullanıcıya özel uyarlanmış metin, görsel, düzen ve hatta tam arayüzler gibi kullanıcı arayüzü öğelerini üretmek için kullanılıp kullanılamayacağıdır. Model, kullanıcının geçmiş etkileşimlerini, belirtilen tercihlerini, demografik bilgilerini ve mevcut kullanım bağlamını dikkate alarak son derece kişiselleştirilmiş ve alakalı arayüzler oluşturmak üzere yönlendirilebilir.

GenUI, geleneksel kullanıcı arayüzü tasarımından birkaç temel açıdan farklılık gösterir:

1. **Dinamik ve Uyarlanabilir**: Geleneksel kullanıcı arayüzü tasarımı, tüm kullanıcılar için aynı kalan sabit, statik arayüzler oluşturmayı içerir. Buna

karşılık, GenUI kullanıcı ihtiyaçlarına ve bağlama göre dinamik olarak uyum sağlayabilen ve değişebilen arayüzler sağlar. Bu, aynı uygulamanın farklı kullanıcılara veya hatta aynı kullanıcıya farklı durumlarda farklı arayüzler sunabileceği anlamına gelir.

2. **Ölçeklenebilir Kişiselleştirme**: Geleneksel tasarımda, gereken zaman ve kaynaklar nedeniyle her kullanıcı için kişiselleştirilmiş deneyimler oluşturmak genellikle pratik değildir. Öte yandan GenUI, ölçeklenebilir kişiselleştirmeye olanak tanır. Yapay zekayı kullanarak tasarımcılar, her kullanıcı segmenti için ayrı arayüzler manuel olarak tasarlayıp geliştirmek zorunda kalmadan, her kullanıcının benzersiz ihtiyaçlarına ve tercihlerine otomatik olarak uyum sağlayan arayüzler oluşturabilirler.
3. **Sonuçlara Odaklanma**: Geleneksel kullanıcı arayüzü tasarımı genellikle görsel açıdan çekici ve işlevsel arayüzler oluşturmaya odaklanır. Bu yönler GenUI'de hala önemli olmakla birlikte, temel odak istenen kullanıcı sonuçlarını elde etmeye kayar. GenUI, salt estetik kaygılardan ziyade kullanılabilirliği ve etkinliği ön planda tutarak, her kullanıcının özel hedeflerine ve görevlerine optimize edilmiş arayüzler oluşturmayı amaçlar.
4. **Sürekli Öğrenme ve İyileştirme**: GenUI sistemleri, kullanıcı etkileşimleri ve geri bildirimlere dayalı olarak sürekli öğrenebilir ve zamanla gelişebilir. Kullanıcılar oluşturulan arayüzlerle etkileşime girdikçe, yapay zeka modelleri kullanıcı davranışları, tercihleri ve sonuçları hakkında veri toplayabilir ve bu bilgileri gelecekteki arayüz oluşturmalarını iyileştirmek ve optimize etmek için kullanabilir. Bu yinelemeli öğrenme süreci, GenUI sistemlerinin zamanla kullanıcı ihtiyaçlarını karşılamada giderek daha etkili hale gelmesini sağlar.

GenUI'nin, belirli tasarım görevlerini otomatikleştiren veya öneriler sunan yapay zeka destekli tasarım araçlarıyla aynı şey olmadığını belirtmek önemlidir. Bu araçlar tasarım sürecini kolaylaştırmada yardımcı olabilse de, yine de nihai kararları vermek ve statik arayüzler oluşturmak için tasarımcılara güvenirler. Öte yandan GenUI,

yapay zeka sisteminin kullanıcı verileri ve bağlama dayalı olarak arayüzlerin gerçek oluşturulmasında ve uyarlanmasında daha aktif bir rol almasını içerir.

GenUI, kullanıcı arayüzü tasarımına yaklaşımımızda, herkese uyan tek tip çözümlerden son derece kişiselleştirilmiş, uyarlanabilir deneyimlere doğru önemli bir değişimi temsil eder. Yapay zekanın gücünden yararlanan GenUI, dijital ürünler ve hizmetlerle etkileşim kurma şeklimizi devrimleştirme potansiyeline sahiptir ve her bir kullanıcı için daha sezgisel, ilgi çekici ve etkili arayüzler oluşturur.

Örnek

GenUI kavramını örneklendirmek için, "FitAI" adlı varsayımsal bir fitness uygulamasını ele alalım. Bu uygulama, kullanıcılara bireysel hedeflerine, fitness seviyelerine ve tercihlerine göre kişiselleştirilmiş antrenman planları ve beslenme tavsiyeleri sunmayı amaçlamaktadır.

Geleneksel bir kullanıcı arayüzü tasarım yaklaşımında, FitAI tüm kullanıcılar için aynı olan sabit bir ekran ve öğe setine sahip olabilir. Ancak GenUI ile uygulamanın arayüzü her kullanıcının benzersiz ihtiyaçlarına ve bağlamına dinamik olarak uyum sağlayabilir.

Bu yaklaşımı 2024'te uygulamayı hayal etmek biraz zorlama olabilir ve hatta yeterli yatırım getirisi sağlamayabilir, ancak mümkündür.

İşte nasıl çalışabileceğine dair bir örnek:

1. **Başlangıç Süreci**:
    - Standart bir anket yerine, FitAI kullanıcının hedefleri, mevcut fitness seviyesi ve tercihleri hakkında bilgi toplamak için konuşma tabanlı yapay zeka kullanır.
    - Bu ilk etkileşime dayanarak, yapay zeka kullanıcının hedefleriyle en alakalı özellikleri ve bilgileri vurgulayan kişiselleştirilmiş bir gösterge paneli düzeni oluşturur.

- Mevcut yapay zeka teknolojisi, kişiselleştirilmiş gösterge panelini oluşturmak için kullanabileceği bir dizi ekran bileşenine sahip olabilir.
- Gelecekteki yapay zeka teknolojisi, deneyimli bir kullanıcı arayüzü tasarımcısı rolünü üstlenebilir ve gösterge panelini *sıfırdan* gerçekten oluşturabilir.

2. **Antrenman Planlayıcısı**:

- Antrenman planlayıcı arayüzü, kullanıcının deneyim seviyesine ve mevcut ekipmanına göre yapay zeka tarafından özel olarak uyarlanır.
- Ekipmanı olmayan bir başlangıç seviyesi kullanıcısı için, detaylı talimatlar ve videolarla basit vücut ağırlığı egzersizleri gösterebilir.
- Spor salonuna erişimi olan ileri seviye bir kullanıcı için, daha az açıklayıcı içerikle daha karmaşık rutinler görüntüleyebilir.
- Antrenman planlayıcısının içeriği basitçe büyük bir üst kümeden filtrelenmez. Kullanıcı hakkında bilinen her şeyi içeren bağlamla sorgulanan bir bilgi tabanına dayanarak *anında* üretilebilir.

3. **İlerleme Takibi**:

- İlerleme takibi arayüzü, kullanıcının hedeflerine ve katılım modellerine göre gelişir.
- Öncelikle kilo vermeye odaklanan bir kullanıcı için, arayüz ağırlık trendi grafiğini ve kalori yakımı istatistiklerini öne çıkarabilir.
- Kas geliştiren bir kullanıcı için, güç artışını ve vücut kompozisyonu değişikliklerini vurgulayabilir.
- Yapay zeka, uygulamanın bu bölümünü kullanıcının gerçek ilerlemesine göre uyarlayabilir. İlerleme bir süre durduğunda, uygulama, bunları hafifletmek amacıyla kullanıcıyı gerilemenin nedenlerini açıklamaya teşvik eden bir moda geçebilir.

4. **Beslenme Tavsiyeleri**:

- Beslenme bölümü kullanıcının diyet tercihlerine ve kısıtlamalarına uyum sağlar.
- Vegan bir kullanıcı için, bitkisel bazlı yemek önerileri ve protein kaynakları gösterebilir.
- Gluten intoleransı olan bir kullanıcı için, önerilerden gluten içeren yiyecekleri otomatik olarak filtreleyecektir.
- Yine, içerik tüm kullanıcılara uyan büyük bir yemek veri kümesinden çekilmez, bunun yerine kullanıcının özel durumu ve kısıtlamalarına göre uyarlanabilen bilgileri içeren bir bilgi tabanından sentezlenir.
- Örneğin, tarifler, kullanıcının fitness seviyesi ve vücut istatistikleri geliştikçe sürekli değişen kalori ihtiyaçlarına uygun malzeme özellikleriyle oluşturulur.

5. **Motivasyonel Öğeler**:

- Uygulamanın motivasyonel içeriği ve bildirimleri, kullanıcının kişilik tipine ve farklı motivasyonel stratejilere verdiği tepkiye göre kişiselleştirilir.
- Bazı kullanıcılar cesaretlendirici mesajlar alırken, diğerleri daha veri odaklı geri bildirimler alabilir.

Bu örnekte, GenUI, FitAI'ın her kullanıcı için son derece özelleştirilmiş bir deneyim yaratmasını sağlayarak, katılımı, memnuniyeti ve fitness hedeflerine ulaşma olasılığını potansiyel olarak artırır. Arayüz öğeleri, içerik ve hatta uygulamanın "kişiliği", her bir kullanıcının ihtiyaç ve tercihlerine en iyi şekilde hizmet etmek için uyarlanır.

Sonuç Odaklı Tasarıma Geçiş

GenUI, kullanıcı arayüzü tasarımına yaklaşımda, belirli arayüz öğeleri oluşturmaya odaklanmaktan daha bütünsel, sonuç odaklı bir yaklaşıma doğru temel bir değişimi temsil eder. Bu değişimin birkaç önemli etkisi vardır:

1. **Kullanıcı Hedeflerine Odaklanma**:

- Tasarımcıların, belirli arayüz bileşenleri yerine kullanıcı hedefleri ve istenen sonuçlar hakkında daha derin düşünmeleri gerekecek.
- Vurgu, kullanıcıların hedeflerine verimli ve etkili bir şekilde ulaşmalarına yardımcı olacak arayüzler üretebilen sistemler oluşturmak üzerine olacak.
- Yapay zeka tabanlı tasarımcılara, önceden tanımlanmış ekran özelliklerini temel almak yerine kullanıcı deneyimlerini *anında* ve *sıfırdan* oluşturma yeteneği veren yeni UI çerçeveleri ortaya çıkacak.

2. **Tasarımcıların Değişen Rolü**:

   - Tasarımcılar, sabit düzenler oluşturmaktan, yapay zeka sistemlerinin arayüzler üretirken takip edeceği kuralları, kısıtlamaları ve yönergeleri tanımlamaya geçiş yapacak.
   - GenUI sistemlerini etkili bir şekilde yönlendirmek için veri analizi, yapay zeka prompt mühendisliği ve sistem düşüncesi gibi alanlarda beceriler geliştirmeleri gerekecek.

3. **Kullanıcı Araştırmasının Önemi**:

   - Kullanıcı araştırması, bir GenUI bağlamında daha da kritik hale geliyor, çünkü tasarımcıların sadece kullanıcı tercihlerini değil, aynı zamanda bu tercih ve ihtiyaçların farklı bağlamlarda nasıl değiştiğini de anlamaları gerekiyor.
   - Sürekli kullanıcı testi ve geri bildirim döngüleri, yapay zekanın etkili arayüzler üretme yeteneğini geliştirmek ve iyileştirmek için gerekli olacak.

4. **Değişkenlik için Tasarım**:

   - Tek bir "mükemmel" arayüz oluşturmak yerine, tasarımcıların çoklu olası varyasyonları düşünmeleri ve sistemin farklı kullanıcı ihtiyaçları için uygun arayüzler üretebilmesini sağlamaları gerekecek.

- Bu, uç durumlar için tasarım yapmayı ve oluşturulan arayüzlerin farklı konfigürasyonlarda kullanılabilirlik ve erişilebilirliği korumasını sağlamayı içerir.
- Ürün farklılaştırması, kullanıcı psikolojisi konusunda farklı bakış açıları ve rakiplerin erişemediği benzersiz veri setleri ve bilgi tabanlarının kullanılmasını içeren yeni boyutlar kazanır.

Zorluklar ve Dikkat Edilmesi Gerekenler

GenUI heyecan verici olanaklar sunarken, aynı zamanda birkaç zorluk ve dikkat edilmesi gereken nokta da ortaya çıkarır:

1. **Teknik Sınırlamalar**:
    - Mevcut yapay zeka teknolojisi, gelişmiş olmasına rağmen, karmaşık kullanıcı niyetlerini anlama ve gerçekten bağlam duyarlı arayüzler üretme konusunda hala sınırlamalara sahiptir.
    - Özellikle daha az güçlü cihazlarda, arayüz öğelerinin gerçek zamanlı üretimiyle ilgili performans sorunları.
2. **Veri Gereksinimleri**:
    - Kullanım senaryosuna bağlı olarak, etkili GenUI sistemleri kişiselleştirilmiş arayüzler oluşturmak için önemli miktarda kullanıcı verisi gerektirebilir.
    - Otantik kullanıcı verilerinin etik bir şekilde elde edilmesindeki zorluklar, veri gizliliği ve güvenliği ile birlikte GenUI modellerini eğitmek için kullanılan verilerdeki olası önyargılar konusunda endişelere yol açmaktadır.
3. **Kullanılabilirlik ve Tutarlılık**:

- En azından bu uygulama yaygınlaşana kadar, sürekli değişen arayüzlere sahip bir uygulama, kullanıcıların tanıdık öğeleri bulmakta veya etkili bir şekilde gezinmekte zorlanması nedeniyle kullanılabilirlik sorunlarına yol açabilir.
- Kişiselleştirme ile tutarlı ve öğrenilebilir bir arayüz arasında denge kurmak çok önemli olacaktır.

4. **Yapay Zekaya Aşırı Güven**:

   - Tasarım kararlarını yapay zeka sistemlerine aşırı devretme riski, ilham vermeyen, sorunlu veya basitçe bozuk arayüz seçimlerine yol açabilir.
   - İnsan gözetimi ve yapay zeka tarafından oluşturulan tasarımları geçersiz kılma yeteneği, öngörülebilir gelecekte önemini koruyacaktır.

5. **Erişilebilirlik Endişeleri**:

   - Dinamik olarak oluşturulan arayüzlerin engelli kullanıcılar için erişilebilir kalmasını sağlamak, tipik sistemlerin gösterdiği düşük erişilebilirlik uyumu göz önüne alındığında endişe verici olan tamamen yeni zorluklar ortaya çıkarır.
   - Öte yandan, yapay zeka tasarımcıları erişilebilirlik konusunda *yerleşik* bir hassasiyetle ve engelli olmayan kullanıcılar için UI oluşturdukları gibi anında erişilebilir arayüzler oluşturma yetenekleriyle uygulanabilir.
   - Her durumda, GenUI sistemleri sağlam erişilebilirlik yönergeleri ve test süreçleriyle tasarlanmalıdır.

6. **Kullanıcı Güveni ve Şeffaflık**:

   - Kullanıcılar, kendileri hakkında "çok fazla şey bilen" veya anlamadıkları şekillerde değişen arayüzlerle rahatsız olabilirler.
   - Arayüzlerin nasıl ve neden kişiselleştirildiği konusunda şeffaflık sağlamak, kullanıcı güvenini oluşturmak için önemli olacaktır.

Gelecek Görünümü ve Fırsatlar

Üretken Kullanıcı Arayüzünün (GenUI) geleceği, dijital ürünler ve hizmetlerle etkileşim şeklimizi devrimselleştirme konusunda muazzam bir potansiyel barındırıyor. Bu teknoloji geliştikçe, kullanıcı arayüzlerinin tasarlanma, uygulanma ve deneyimlenme şeklinde büyük bir değişim bekleyebiliriz. GenUI'nin, yazılımlarımızı şu anda bilim kurgu olarak kabul edilen alana taşıyacak olgu olduğunu düşünüyorum.

GenUI'nin en heyecan verici yönlerinden biri, erişilebilirliği ciddi engelli insanların yazılımınızı kullanmaktan tamamen dışlanmamasını sağlamanın ötesine geçen büyük bir ölçekte artırma potansiyelidir. Arayüzleri otomatik olarak bireysel kullanıcı ihtiyaçlarına uyarlayarak, GenUI dijital deneyimleri her zamankinden daha kapsayıcı hale getirebilir. Manuel yapılandırma veya uygulamaların ayrı "erişilebilir" sürümlerini gerektirmeden, genç veya görme engelli kullanıcılar için daha büyük metin sağlayan veya bilişsel engelli olanlar için basitleştirilmiş düzenler sunan arayüzleri hayal edin.

GenUI'nin kişiselleştirme yetenekleri, çeşitli dijital ürünlerde kullanıcı katılımını, memnuniyetini ve sadakatini artıracak gibi görünüyor. Arayüzler bireysel tercihler ve davranışlara daha uyumlu hale geldikçe, kullanıcılar dijital deneyimleri daha sezgisel ve keyifli bulacak, bu da teknolojiyle daha derin ve anlamlı etkileşimlere yol açabilecek.

GenUI ayrıca yeni kullanıcılar için başlangıç sürecini dönüştürme potansiyeline sahip. Her kullanıcının uzmanlık düzeyine hızla uyum sağlayan sezgisel, kişiselleştirilmiş ilk kullanıcı deneyimleri oluşturarak, yeni uygulamalarla ilişkili öğrenme sürecini önemli ölçüde azaltabilir. Bu, daha hızlı benimseme oranlarına ve yeni özellikleri ve işlevleri keşfetme konusunda artan kullanıcı güvenine yol açabilir.

Bir diğer heyecan verici olasılık, GenUI'nin her özel kullanım bağlamı için optimize ederken farklı cihazlar ve platformlar arasında tutarlı bir kullanıcı deneyimini sürdürme yeteneğidir. Bu, akıllı telefonlardan tabletlere, masaüstü bilgisayarlara ve artırılmış gerçeklik gözlükleri gibi gelişen teknolojilere kadar giderek parçalanan cihaz ortamında tutarlı deneyimler sağlama konusundaki uzun süredir devam eden zorluğu çözebilir.

GenUI'nin veri odaklı doğası, kullanıcı arayüzü tasarımında hızlı yineleme ve iyileştirme fırsatları sunar. Kullanıcıların oluşturulan arayüzlerle nasıl etkileşime girdiğine dair gerçek zamanlı veriler toplayarak, tasarımcılar ve geliştiriciler kullanıcı davranışı ve tercihleri hakkında benzeri görülmemiş içgörüler elde edebilir. Bu geri bildirim döngüsü, varsayımlar veya sınırlı kullanıcı testleri yerine gerçek kullanım kalıplarına dayalı olarak UI tasarımında sürekli iyileştirmelere yol açabilir.

Bu değişime hazırlanmak için, tasarımcıların beceri setlerini ve düşünce biçimlerini geliştirmeleri gerekecek. Odak noktası, sabit düzenler oluşturmaktan, yapay zeka destekli arayüz oluşturmaya rehberlik edebilecek kapsamlı tasarım sistemleri ve yönergeler geliştirmeye kayacak. Tasarımcılar, GenUI sistemlerini etkili bir şekilde yönlendirmek için veri analizi, yapay zeka teknolojileri ve sistem düşüncesi konularında derin bir anlayış geliştirmeleri gerekecek.

Dahası, GenUI tasarım ve teknoloji arasındaki çizgileri bulanıklaştırdıkça, tasarımcıların geliştiriciler ve veri bilimcileriyle daha yakın işbirliği yapması gerekecek. Bu disiplinler arası yaklaşım, sadece görsel olarak çekici ve kullanıcı dostu değil, aynı zamanda teknik olarak sağlam ve etik açıdan uygun GenUI sistemleri oluşturmada çok önemli olacaktır.

Teknoloji olgunlaştıkça GenUI'nin etik sonuçları da ön plana çıkacaktır. Tasarımcılar, arayüz tasarımında yapay zekânın sorumlu kullanımı için çerçeveler geliştirmede kritik bir rol oynayacak ve kişiselleştirmenin gizliliği tehlikeye atmadan veya kullanıcı davranışlarını etik olmayan şekillerde manipüle etmeden kullanıcı deneyimlerini geliştirmesini sağlayacaktır.

Geleceğe baktığımızda, GenUI hem heyecan verici fırsatlar hem de önemli zorluklar sunmaktadır. Dünya çapındaki kullanıcılar için daha sezgisel, verimli ve tatmin edici dijital deneyimler yaratma potansiyeline sahiptir. Tasarımcıların uyum sağlamasını ve yeni beceriler edinmesini gerektirse de, aynı zamanda insan-bilgisayar etkileşiminin geleceğini derin ve anlamlı yollarla şekillendirmek için eşsiz bir fırsat sunmaktadır. Tam anlamıyla gerçekleşmiş GenUI sistemlerine giden yolculuk kuşkusuz karmaşık olacaktır, ancak gelişmiş kullanıcı deneyimleri ve dijital erişilebilirlik açısından

potansiyel kazanımlar, uğrunda çaba gösterilmeye değer bir geleceği işaret etmektedir.

Akıllı İş Akışı Orkestrayonu

“Akıllı İş Akışı Orkestrayonu” yaklaşımı, uygulamalar içindeki karmaşık iş akışlarını dinamik olarak orkestra etmek ve optimize etmek için yapay zeka bileşenlerinden yararlanmaya odaklanır. Amaç, daha verimli, duyarlı ve gerçek zamanlı verilerle bağlama uyum sağlayabilen uygulamalar oluşturmaktır.

Bu bölümde, akıllı iş akışı orkestrayonu yaklaşımının temelini oluşturan ana ilkeleri ve kalıpları inceleyeceğiz. Yapay zekanın görevleri akıllıca yönlendirmek, karar vermeyi otomatikleştirmek ve iş akışlarını kullanıcı davranışı, sistem performansı ve iş kuralları gibi çeşitli faktörlere göre dinamik olarak uyarlamak için nasıl kullanılabileceğini ele alacağız. Pratik örnekler ve gerçek dünya senaryoları aracılığıyla, yapay zekanın uygulama iş akışlarını düzenlemek ve optimize etmedeki dönüştürücü potansiyelini göstereceğiz.

İster karmaşık iş süreçlerine sahip kurumsal uygulamalar, ister dinamik kullanıcı yolculukları olan tüketici odaklı uygulamalar geliştiriyor olun, bu bölümde tartışılan

kalıplar ve teknikler, genel kullanıcı deneyimini geliştiren ve iş değeri yaratan akıllı ve verimli iş akışları oluşturmanız için gereken bilgi ve araçlarla sizi donatacaktır.

İş İhtiyacı

İş akışı yönetimine yönelik geleneksel yaklaşımlar genellikle önceden tanımlanmış kurallara ve statik karar ağaçlarına dayanır; bu da katı, esnek olmayan ve modern uygulamaların dinamik doğasıyla başa çıkamayan sonuçlar doğurabilir.

Bir e-ticaret uygulamasının karmaşık bir sipariş karşılama sürecini yönetmesi gereken bir senaryoyu düşünün. İş akışı, sipariş doğrulama, stok kontrolü, ödeme işleme, kargo ve müşteri bildirimleri gibi birden çok adım içerebilir. Her adımın kendi kural setleri, bağımlılıkları, dış entegrasyonları ve istisna yönetimi mekanizmaları olabilir. Böyle bir iş akışını manuel olarak veya sabit kodlanmış mantıkla yönetmek hızla zahmetli, hataya açık ve bakımı zor hale gelebilir.

Dahası, uygulama ölçeklendikçe ve eşzamanlı kullanıcı sayısı arttıkça, iş akışının gerçek zamanlı verilere ve sistem performansına göre kendini uyarlaması ve optimize etmesi gerekebilir. Örneğin, yoğun trafik dönemlerinde uygulamanın, belirli görevlere öncelik vermek, kaynakları verimli bir şekilde tahsis etmek ve sorunsuz bir kullanıcı deneyimi sağlamak için iş akışını dinamik olarak ayarlaması gerekebilir.

İşte "Akıllı İş Akışı Orkestrayonu" yaklaşımı burada devreye girer. Yapay zeka bileşenlerinden yararlanarak, geliştiriciler akıllı, uyarlanabilir ve kendi kendini optimize eden iş akışları oluşturabilir. Yapay zeka, büyük miktarda veriyi analiz edebilir, geçmiş deneyimlerden öğrenebilir ve iş akışını etkili bir şekilde yönetmek için gerçek zamanlı bilgiye dayalı kararlar alabilir.

Temel Faydalar

1. **Artan Verimlilik**: Yapay zeka, görev tahsisini, kaynak kullanımını ve iş akışı yürütmesini optimize ederek daha hızlı işlem süreleri ve gelişmiş genel verimlilik

sağlayabilir.

2. **Uyarlanabilirlik**: Yapay zeka destekli iş akışları, kullanıcı talebindeki dalgalanmalar, sistem performansı veya iş gereksinimleri gibi değişen koşullara dinamik olarak uyum sağlayarak uygulamanın duyarlı ve dirençli kalmasını sağlar.
3. **Otomatik Karar Verme**: Yapay zeka, iş akışı içindeki karmaşık karar verme süreçlerini otomatikleştirebilir, manuel müdahaleyi azaltabilir ve insan hatası riskini en aza indirebilir.
4. **Kişiselleştirme**: Yapay zeka, kullanıcı davranışını, tercihlerini ve bağlamı analiz ederek iş akışını kişiselleştirebilir ve bireysel kullanıcılara özelleştirilmiş deneyimler sunabilir.
5. **Ölçeklenebilirlik**: Yapay zeka destekli iş akışları, performans veya güvenilirlikten ödün vermeden artan veri ve kullanıcı etkileşimi hacmini yönetmek için sorunsuz bir şekilde ölçeklenebilir.

Sonraki bölümlerde, akıllı iş akışlarının uygulanmasını sağlayan temel kalıpları ve teknikleri inceleyecek ve yapay zekanın modern uygulamalardaki iş akışı yönetimini nasıl dönüştürdüğünü gösteren gerçek dünya örneklerini sunacağız.

Temel Kalıplar

Uygulamalarda akıllı iş akışı orkestrayonunu uygulamak için geliştiriciler, yapay zekanın gücünden yararlanan birkaç temel kalıptan faydalanabilir. Bu kalıplar, uygulamaların gerçek zamanlı verilere ve bağlama dayalı olarak süreçleri uyarlaması, optimize etmesi ve otomatikleştirmesi için iş akışlarını tasarlama ve yönetmede yapılandırılmış bir yaklaşım sunar. Akıllı iş akışı orkestrayonundaki temel kalıplardan bazılarını inceleyelim.

Dinamik Görev Yönlendirme

Bu kalıp, görev önceliği, kaynak kullanılabilirliği ve sistem performansı gibi çeşitli faktörlere dayalı olarak bir iş akışı içindeki görevleri akıllıca yönlendirmek için yapay zeka kullanımını içerir. Yapay zeka algoritmaları her görevin özelliklerini analiz edebilir, sistemin mevcut durumunu değerlendirebilir ve görevleri en uygun kaynaklara veya işlem yollarına atamak için bilgiye dayalı kararlar verebilir. Dinamik görev yönlendirme, görevlerin verimli bir şekilde dağıtılmasını ve yürütülmesini sağlayarak genel iş akışı performansını optimize eder.

```
class TaskRouter
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  attr_accessor :task

  # list of functions that can be called by the AI entirely at its
  # discretion depending on the task received

  function :analyze_task_priority do
    TaskPriorityAnalyzer.perform(task)
  end

  function :check_resource_availability, # ...
  function :assess_system_performance, # ...
  function :assign_task_to_resource, # ...

  DIRECTIVE = "You are a task router, responsible for intelligently
   assigning tasks to available resources based on priority, resource
   availability, and system performance..."

  def initialize(task)
    self.task = task
    transcript << { system: DIRECTIVE }
    transcript << { user: task.to_json }
  end

  def perform
    while task.unassigned?
```

```
30        chat_completion
31
32        # todo: add max loop counter and break
33      end
34
35      # capture the transcript for later analysis
36      task.update(routing_transcript: transcript)
37    end
38  end
```

29. satırdaki `while` ifadesi ile oluşturulan döngüye dikkat edin; bu döngü görev atanana kadar yapay zekâya sorgu göndermeye devam eder. 35. satırda, gerektiğinde daha sonra analiz ve hata ayıklama için görevin dökümünü kaydediyoruz.

Bağlamsal Karar Verme

Bir iş akışı içinde bağlama duyarlı kararlar vermek için çok benzer bir kod kullanabilirsiniz. Kullanıcı tercihleri, geçmiş örüntüler ve gerçek zamanlı girdiler gibi ilgili veri noktalarını analiz ederek, yapay zekâ bileşenleri iş akışındaki her karar noktasında en uygun eylem şeklini belirleyebilir. İş akışınızın davranışını her kullanıcının veya senaryonun özel bağlamına göre uyarlayarak, kişiselleştirilmiş ve optimize edilmiş deneyimler sunabilirsiniz.

Uyarlanabilir İş Akışı Kompozisyonu

Bu örüntü, değişen gereksinimler veya koşullara göre iş akışlarını dinamik olarak oluşturmaya ve ayarlamaya odaklanır. Yapay zekâ, iş akışının mevcut durumunu analiz edebilir, darboğazları veya verimsizlikleri tespit edebilir ve performansı optimize etmek için iş akışı yapısını otomatik olarak değiştirebilir. Uyarlanabilir iş akışı kompozisyonu, uygulamaların manuel müdahale gerektirmeden sürekli olarak gelişmesine ve süreçlerini iyileştirmesine olanak tanır.

İstisna İşleme ve Kurtarma

İstisna işleme ve kurtarma, akıllı iş akışı orkestrasyon sürecinin kritik yönleridir. Yapay zekâ bileşenleri ve karmaşık iş akışlarıyla çalışırken, sistemin istikrarını ve güvenilirliğini sağlamak için istisnaları öngörmek ve bunları zarif bir şekilde ele almak esastır.

Akıllı iş akışlarında istisna işleme ve kurtarma için bazı önemli hususlar ve teknikler şunlardır:

1. **İstisna Yayılımı**: İş akışı bileşenleri arasında istisnaların yayılması için tutarlı bir yaklaşım uygulayın. Bir bileşen içinde bir istisna oluştuğunda, bu istisna yakalanmalı, kaydedilmeli ve orkestratöre veya istisnaları işlemekten sorumlu ayrı bir bileşene iletilmelidir. Amaç, istisna işlemeyi merkezileştirmek ve istisnaların sessizce yutulmasını önlemek ve aynı zamanda Akıllı Hata İşleme için olanaklar yaratmaktır.
2. **Yeniden Deneme Mekanizmaları**: Yeniden deneme mekanizmaları, iş akışının dayanıklılığını artırmaya ve geçici hataları zarif bir şekilde ele almaya yardımcı olur. Ağ bağlantısı veya kaynak kullanılamaması gibi geçici veya kurtarılabilir istisnalar için belirli bir gecikmeden sonra otomatik olarak yeniden denenebilecek yeniden deneme mekanizmalarını mutlaka uygulamaya çalışın. Yapay zekâ destekli bir orkestratör veya istisna işleyiciye sahip olmak, yeniden deneme stratejilerinizin üstel geri çekilme gibi sabit algoritmalara dayanmasını gerektirmez. İstisnayı nasıl ele alacağına karar vermekten sorumlu yapay zekâ bileşeninin "takdirine" bırakabilirsiniz.
3. **Geri Dönüş Stratejileri**: Bir yapay zekâ bileşeni geçerli bir yanıt sağlayamazsa veya bir hatayla karşılaşırsa—ki bu, öncü teknoloji doğası gereği sık karşılaşılan bir durumdur—iş akışının devam edebilmesi için bir geri dönüş mekanizması bulundurun. Bu, varsayılan değerleri kullanmayı, alternatif algoritmaları veya kararlar almak ve iş akışını ilerletmek için Döngüde İnsan kullanmayı içerebilir.

4. **Telafi Edici Eylemler**: Orkestratörün yönergeleri, otomatik olarak çözülemeyen istisnaları ele almak için telafi edici eylemler hakkında talimatlar içermelidir. Telafi edici eylemler, başarısız olan bir işlemin etkilerini geri almak veya azaltmak için atılan adımlardır. Örneğin, bir ödeme işleme adımı başarısız olursa, telafi edici eylem işlemi geri almak ve kullanıcıyı bilgilendirmek olabilir. Telafi edici eylemler, istisnalar karşısında veri tutarlılığını ve bütünlüğünü korumaya yardımcı olur.
5. **İstisna İzleme ve Uyarı Verme**: Kritik istisnaları tespit etmek ve ilgili paydaşları bilgilendirmek için izleme ve uyarı verme mekanizmaları kurun. Orkestratör, istisnalar belirli sınırları aştığında veya belirli türde istisnalar oluştuğunda uyarıları tetikleyecek eşiklerden ve kurallardan haberdar edilebilir. Bu, sorunların genel sistemi etkilemeden önce proaktif olarak tanımlanmasına ve çözülmesine olanak tanır.

İşte Ruby iş akışı bileşeninde istisna işleme ve kurtarmanın bir örneği:

```ruby
class InventoryManager
  def check_availability(order)
    begin
      # Perform inventory check logic
      inventory = Inventory.find_by(product_id: order.product_id)
      if inventory.available_quantity >= order.quantity
        return true
      else
        raise InsufficientInventoryError,
              "Insufficient inventory for product #{order.product_id}"
      end
    rescue InsufficientInventoryError => e
      # Log the exception
      logger.error("Inventory check failed: #{e.message}")

      # Retry the operation after a delay
      retry_count ||= 0
      if retry_count < MAX_RETRIES
        retry_count + = 1
        sleep(RETRY_DELAY)
```

```
        retry
      else
        # Fallback to manual intervention
        NotificationService.admin("Inventory check failed: Order #{order.id}")
        return false
      end
    end
  end
end
```

Bu örnekte, `InventoryManager` bileşeni belirli bir sipariş için ürün mevcudiyetini kontrol eder. Eğer mevcut miktar yetersizse, bir `InsufficientInventoryError` istisnası fırlatır. İstisna yakalanır, kaydedilir ve yeniden deneme mekanizması uygulanır. Yeniden deneme sınırı aşılırsa, bileşen bir yöneticiyi bilgilendirerek manuel müdahaleye geçer.

Sağlam istisna yönetimi ve kurtarma mekanizmaları uygulayarak, akıllı iş akışlarınızın dayanıklı, sürdürülebilir ve beklenmeyen durumları zarif bir şekilde ele alabileceğinden emin olabilirsiniz.

Bu kalıplar, akıllı iş akışı orkestrasyonunun temelini oluşturur ve farklı uygulamaların özel gereksinimlerine uyacak şekilde birleştirilebilir ve uyarlanabilir. Geliştiriciler bu kalıpları kullanarak esnek, dayanıklı ve performans ile kullanıcı deneyimi için optimize edilmiş iş akışları oluşturabilirler.

Bir sonraki bölümde, bu kalıpların pratikte nasıl uygulanabileceğini, gerçek dünya örnekleri ve kod parçacıkları kullanarak AI bileşenlerinin iş akışı yönetimine entegrasyonunu göstereceğiz.

Akıllı İş Akışı Orkestrasyonunun Pratikte Uygulanması

Akıllı iş akışı orkestrasyonundaki temel kalıpları incelediğimize göre, şimdi bu kalıpların gerçek dünya uygulamalarında nasıl uygulanabileceğine bakalım. AI bileşenlerinin iş akışı yönetimine entegrasyonunu göstermek için pratik örnekler ve kod parçacıkları sunacağız.

Akıllı Sipariş İşleyici

Ruby on Rails e-ticaret uygulamasında AI destekli bir `OrderProcessor` bileşeni kullanarak akıllı iş akışı orkestrasyonunun pratik bir örneğine dalalım. `OrderProcessor`, Bölüm 3'te Çoklu İşçiler konusunu tartışırken ilk karşılaştığımız Süreç Yöneticisi Kurumsal Entegrasyonu kavramını gerçekleştirir. Bu bileşen, sipariş karşılama iş akışını yönetmekten, ara sonuçlara dayalı yönlendirme kararları vermekten ve çeşitli işlem adımlarının yürütülmesini orkestra etmekten sorumlu olacaktır.

Sipariş karşılama süreci, sipariş doğrulama, envanter kontrolü, ödeme işleme ve sevkiyat gibi birden fazla adım içerir. Her adım, belirli bir görevi gerçekleştiren ve sonucu `OrderProcessor`'a döndüren ayrı bir işçi süreci olarak uygulanır. Bu adımlar zorunlu değildir ve hatta kesin bir sırayla yapılmaları gerekmez.

İşte `OrderProcessor`'ın bir örnek uygulaması. Raix'ten iki mixin özelliği içerir. Birincisi (`ChatCompletion`) sohbet tamamlama yapabilme yeteneği verir, bu da onu bir AI bileşeni yapar. İkincisi (`FunctionDispatch`) AI tarafından fonksiyon çağırmayı mümkün kılar, böylece bir komuta metin mesajı yerine fonksiyon çağrısıyla yanıt verebilir.

İşçi fonksiyonları (`validate_order`, `check_inventory`, vb.) ilgili işçi sınıflarına yetki devreder. Bu sınıflar AI veya AI olmayan bileşenler olabilir; tek gereklilikleri, çalışmalarının sonuçlarını bir dizge olarak temsil edilebilecek bir formatta döndürmeleridir.

Bu kitabın bu bölümündeki diğer tüm örneklerde olduğu gibi, bu kod pratik olarak sözde koddur ve yalnızca kalıbın anlamını iletmek ve kendi yaratımlarınıza ilham vermek için tasarlanmıştır. Kalıpların tam açıklamaları ve eksiksiz kod örnekleri Bölüm 2'de yer almaktadır.

```
class OrderProcessor
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."

  def initialize(order)
    self.order = order
    transcript << { system: SYSTEM_DIRECTIVE }
    transcript << { user: order.to_json }
  end

  def perform
    # will continue looping until `stop_looping!` is called
    chat_completion(loop: true)
  end

  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "validate_order",
        description: "Invoke to check validity of order",
        parameters: {
          ...
      },
      ...
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

```

```
  def validate_order
    OrderValidationWorker.perform(@order)
  end

  def check_inventory
    InventoryCheckWorker.perform(@order)
  end

  def process_payment
    PaymentProcessingWorker.perform(@order)
  end

  def schedule_shipping
    ShippingSchedulerWorker.perform(@order)
  end

  def send_confirmation
    OrderConfirmationWorker.perform(@order)
  end

  def finished_processing
    @order.update!(transcript:, processed_at: Time.current)
    stop_looping!
  end
end
```

Örnekte, OrderProcessor bir sipariş nesnesiyle başlatılır ve büyük dil modellerine özgü tipik konuşma kaydı formatında iş akışı yürütmesinin dökümünü tutar. Sipariş doğrulama, envanter kontrolü, ödeme işleme ve kargo gibi çeşitli işlem adımlarının yürütülmesini düzenlemek için yapay zekaya tam kontrol verilir.

`chat_completion` metodu her çağrıldığında, yapay zekaya bir fonksiyon çağrısı olarak tamamlama sağlaması için döküm gönderilir. Önceki adımın sonucunu analiz etmek ve uygun eylemi belirlemek tamamen yapay zekanın kontrolündedir. Örneğin, envanter kontrolü düşük stok seviyelerini gösterirse, `OrderProcessor` bir yenileme görevi planlayabilir. Ödeme işlemi başarısız olursa, yeniden deneme başlatabilir veya müşteri desteğini bilgilendirebilir.

> Yukarıdaki örnekte yenileme veya müşteri desteğini bilgilendirme için tanımlanmış fonksiyonlar yok, ancak kesinlikle olabilirdi.

Döküm, her fonksiyon çağrıldığında büyür ve her adımın sonuçları ile yapay zeka tarafından oluşturulan sonraki adımlar için talimatları içeren iş akışı yürütmesinin kaydı olarak işlev görür. Bu döküm, hata ayıklama, denetim ve sipariş karşılama sürecine görünürlük sağlamak için kullanılabilir.

`OrderProcessor`'da yapay zekayı kullanarak, e-ticaret uygulaması iş akışını gerçek zamanlı verilere göre dinamik olarak uyarlayabilir ve istisnaları akıllıca ele alabilir. Yapay zeka bileşeni, bilgiye dayalı kararlar alabilir, iş akışını optimize edebilir ve karmaşık senaryolarda bile sorunsuz sipariş işlemeyi sağlayabilir.

Çalışan işlemlerin tek gereksiniminin, yapay zekanın bir sonraki adımda ne yapacağına karar verirken değerlendireceği anlaşılır bir çıktı döndürmek olması gerçeği, bu yaklaşımın farklı sistemleri birbirleriyle entegre ederken tipik olarak gereken girdi/çıktı eşleme çalışmasını nasıl azaltabileceğini fark etmenizi sağlayabilir.

Akıllı İçerik Denetleyici

Sosyal medya uygulamaları, genellikle güvenli ve sağlıklı bir topluluk sağlamak için en azından minimal düzeyde içerik denetimine ihtiyaç duyar. Bu örnek `ContentModerator` bileşeni, içeriğin özelliklerine ve çeşitli denetim adımlarının sonuçlarına dayalı olarak kararlar alarak denetim iş akışını akıllıca düzenlemek için yapay zekadan yararlanır.

Denetim süreci, metin analizi, görüntü tanıma, kullanıcı itibarı değerlendirmesi ve manuel inceleme gibi birden çok adım içerir. Her adım, belirli bir görevi yerine getiren ve sonucu `ContentModerator`'e döndüren ayrı bir çalışan işlem olarak uygulanır.

İşte `ContentModerator`'ün örnek bir uygulaması:

```ruby
class ContentModerator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a content moderator process manager,
    tasked with the workflow involved in moderating user-generated content..."

  def initialize(content)
    @content = content
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: content.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "analyze_text",
        # ...
      },
      {
        name: "recognize_image",
        description: "Invoke to describe images...",
        # ...
      },
      {
        name: "assess_user_reputation",
        # ...
      },
      {
```

```
        name: "escalate_to_manual_review",
        # ...
      },
      {
        name: "approve_content",
        # ...
      },
      {
        name: "reject_content",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_text
    result = TextAnalysisWorker.perform(@content)
    continue_with(result)
  end

  def recognize_image
    result = ImageRecognitionWorker.perform(@content)
    continue_with(result)
  end

  def assess_user_reputation
    result = UserReputationWorker.perform(@content.user)
    continue_with(result)
  end

  def escalate_to_manual_review
    ManualReviewWorker.perform(@content)
    @content.update!(status: 'pending', transcript: @transcript)
  end

  def approve_content
    @content.update!(status: 'approved', transcript: @transcript)
  end

  def reject_content
```

```
    @content.update!(status: 'rejected', transcript: @transcript)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

Bu örnekte, `ContentModerator` bir içerik nesnesiyle başlatılır ve konuşma formatında bir moderasyon kaydı tutar. Yapay zeka bileşeni, moderasyon iş akışı üzerinde tam kontrole sahiptir ve içeriğin özelliklerine ve her adımın sonuçlarına göre hangi adımların yürütüleceğine karar verir.

Yapay zekanın çağırabileceği mevcut işçi fonksiyonları arasında `analyze_text`, `recognize_image`, `assess_user_reputation` ve `escalate_to_manual_review` bulunur. Her fonksiyon görevi ilgili işçi sürecine (`TextAnalysisWorker`, `ImageRecognitionWorker`, vb.) devreder ve yükseltme fonksiyonu hariç sonucu moderasyon kaydına ekler; yükseltme fonksiyonu bir son durum olarak işlev görür. Son olarak, `approve_content` ve `reject_content` fonksiyonları da son durumlar olarak işlev görür.

Yapay zeka bileşeni içeriği analiz eder ve uygun eylemi belirlemeye karar verir. İçerik görsel referanslar içeriyorsa, görsel inceleme için `recognize_image` işçisini çağırabilir. Herhangi bir işçi potansiyel olarak zararlı içerik konusunda uyarıda bulunursa, yapay zeka içeriği manuel incelemeye yükseltmeye veya doğrudan reddetmeye karar verebilir. Ancak uyarının ciddiyetine bağlı olarak, yapay zeka emin olmadığı içeriği nasıl ele alacağına karar verirken kullanıcı itibar değerlendirmesinin sonuçlarını kullanmayı tercih edebilir. Kullanım senaryosuna bağlı olarak, güvenilir kullanıcılar paylaşabilecekleri içerik konusunda daha fazla esnekliğe sahip olabilir. Ve bu böyle devam eder...

Önceki süreç yöneticisi örneğinde olduğu gibi, moderasyon kaydı, her adımın

sonuçlarını ve yapay zeka tarafından üretilen kararları içeren iş akışı yürütmesinin bir kaydı olarak işlev görür. Bu kayıt, denetleme, şeffaflık ve moderasyon sürecini zaman içinde iyileştirmek için kullanılabilir.

`ContentModerator`'de yapay zekayı kullanarak, sosyal medya uygulaması moderasyon iş akışını içeriğin özelliklerine göre dinamik olarak uyarlayabilir ve karmaşık moderasyon senaryolarını akıllıca ele alabilir. Yapay zeka bileşeni bilinçli kararlar alabilir, iş akışını optimize edebilir ve güvenli ve sağlıklı bir topluluk deneyimi sağlayabilir.

Akıllı iş akışı orkestrayonu bağlamında öngörüsel görev planlaması ve istisna yönetimi ve kurtarmayı gösteren iki örneği daha inceleyelim.

Bir Müşteri Destek Sisteminde Öngörüsel Görev Planlaması

Ruby on Rails ile oluşturulmuş bir müşteri destek uygulamasında, destek taleplerinin verimli bir şekilde yönetilmesi ve önceliklendirilmesi, müşterilere zamanında yardım sağlamak için çok önemlidir. `SupportTicketScheduler` bileşeni, destek taleplerini talep aciliyeti, temsilci uzmanlığı ve iş yükü gibi çeşitli faktörlere dayanarak mevcut temsilcilere öngörüsel olarak planlamak ve atamak için yapay zekadan yararlanır.

```ruby
class SupportTicketScheduler
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
    tasked with intelligently assigning tickets to available agents..."

  def initialize(ticket)
    @ticket = ticket
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: ticket.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "analyze_ticket_urgency",
        # ...
      },
      {
        name: "list_available_agents",
        description: "Includes expertise of available agents",
        # ...
      },
      {
        name: "predict_agent_workload",
        description: "Uses historical data to predict upcoming workloads",
        # ...
      },
      {
        name: "assign_ticket_to_agent",
        # ...
```

```
      },
      {
        name: "reschedule_ticket",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_ticket_urgency
    result = TicketUrgencyAnalyzer.perform(@ticket)
    continue_with(result)
  end

  def list_available_agents
    result = ListAvailableAgents.perform
    continue_with(result)
  end

  def predict_agent_workload
    result = AgentWorkloadPredictor.perform
    continue_with(result)
  end

  def assign_ticket_to_agent
    TicketAssigner.perform(@ticket, @transcript)
  end

  def delay_assignment(until)
    until = DateTimeStandardizer.process(until)
    SupportTicketScheduler.delay(@ticket, @transcript, until)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

Bu örnekte, `SupportTicketScheduler` bir destek bileti nesnesiyle başlatılır ve bir planlama kaydı tutar. AI bileşeni, bilet detaylarını analiz eder ve bilet aciliyeti, temsilci uzmanlığı ve öngörülen temsilci iş yükü gibi faktörlere dayanarak bilet atamasını öngörüsel olarak planlar.

AI'nin çağırabileceği mevcut fonksiyonlar arasında `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload` ve `assign_ticket_to_agent` bulunur. Her fonksiyon, görevi ilgili analizci veya tahmin edici bileşene devreder ve sonucu planlama kaydına ekler. AI ayrıca `delay_assignment` fonksiyonunu kullanarak atamayı erteleme seçeneğine sahiptir.

AI bileşeni, planlama kaydını inceler ve bilet ataması konusunda bilinçli kararlar verir. Bileti işlemek için en uygun temsilciyi belirlerken biletin aciliyetini, mevcut temsilcilerin uzmanlığını ve her temsilcinin öngörülen iş yükünü değerlendirir.

Öngörüsel görev planlamasından yararlanarak, müşteri destek uygulaması bilet atamalarını optimize edebilir, yanıt sürelerini azaltabilir ve genel müşteri memnuniyetini artırabilir. Destek biletlerinin proaktif ve verimli yönetimi, doğru biletlerin doğru temsilcilere doğru zamanda atanmasını sağlar.

Veri İşleme Hattında İstisna Yönetimi ve Kurtarma

İstisnaların yönetilmesi ve hatalardan kurtulma, veri bütünlüğünü sağlamak ve veri kaybını önlemek için hayati önem taşır. `DataProcessingOrchestrator` bileşeni, bir veri işleme hattında istisnaları akıllıca yönetmek ve kurtarma sürecini orkestra etmek için AI'dan yararlanır.

```
class DataProcessingOrchestrator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."

  def initialize(data_batch)
    @data_batch = data_batch
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: data_batch.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "validate_data",
        # ...
      },
      {
        name: "process_data",
        # ...
      },
      {
        name: "request_fix",
        # ...
      },
      {
        name: "retry_processing",
        # ...
      },
      {
        name: "mark_data_as_failed",
```

```
        # ...
      },
      {
        name: "finished",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_data
    result = DataValidator.perform(@data_batch)
    continue_with(result)
  rescue ValidationException => e
    handle_validation_exception(e)
  end

  def process_data
    result = DataProcessor.perform(@data_batch)
    continue_with(result)
  rescue ProcessingException => e
    handle_processing_exception(e)
  end

  def request_fix(description_of_fix)
    result = SmartDataFixer.new(description_of_fix, @data_batch)
    continue_with(result)
  end

  def retry_processing(timeout_in_seconds)
    wait(timeout_in_seconds)
    process_data
  end

  def mark_data_as_failed
    @data_batch.update!(status: 'failed', transcript: @transcript)
  end

  def finished
    @data_batch.update!(status: 'finished', transcript: @transcript)
```

```
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end

  def handle_validation_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end

  def handle_processing_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end
end
```

Bu örnekte, `DataProcessingOrchestrator` bir veri yığını nesnesiyle başlatılır ve bir işleme kaydı tutar. Yapay zeka bileşeni, veri işleme hattını orkestra eder, istisnaları yönetir ve gerektiğinde hatalardan kurtulur.

Yapay zekanın çağırabileceği mevcut fonksiyonlar arasında `validate_data`, `process_data`, `request_fix`, `retry_processing` ve `mark_data_as_failed` bulunur. Her fonksiyon, görevi ilgili veri işleme bileşenine devreder ve sonucu veya istisna ayrıntılarını işleme kaydına ekler.

`validate_data` adımı sırasında bir doğrulama istisnası oluşursa, `handle_validation_exception` fonksiyonu istisna verilerini kayda ekler ve kontrolü yapay zekaya geri verir. Benzer şekilde, `process_data` adımı sırasında bir işleme istisnası oluşursa, yapay zeka kurtarma stratejisine karar verebilir.

Karşılaşılan istisnanın doğasına bağlı olarak, yapay zeka kendi takdiriyle yapay zeka destekli `SmartDataFixer` bileşenine delege eden `request_fix`'i çağırmaya karar verebilir (Kendi Kendini Onaran Veri bölümüne bakın). Veri düzeltici, işlemenin yeniden

denenebilmesi için `@data_batch`'i nasıl değiştirmesi gerektiğine dair açık bir Türkçe açıklama alır. Belki başarılı bir yeniden deneme, doğrulamada başarısız olan kayıtların veri yığınından kaldırılmasını ve/veya bunların insan incelemesi için farklı bir işleme hattına kopyalanmasını içerebilir? Olasılıklar neredeyse sonsuzdur.

Yapay zeka destekli istisna yönetimi ve kurtarmayı dahil ederek, veri işleme uygulaması daha dayanıklı ve hata toleranslı hale gelir. `DataProcessingOrchestrator` akıllıca istisnaları yönetir, veri kaybını en aza indirir ve veri işleme iş akışının sorunsuz yürütülmesini sağlar.

İzleme ve Günlük Tutma

İzleme ve günlük tutma, yapay zeka destekli iş akışı bileşenlerinin ilerlemesi, performansı ve sağlığı hakkında görünürlük sağlar ve geliştiricilerin sistemin davranışını takip edip analiz etmesine olanak tanır. Etkili izleme ve günlük tutma mekanizmalarının uygulanması, akıllı iş akışlarının hata ayıklaması, denetlenmesi ve sürekli iyileştirilmesi için gereklidir.

İş Akışı İlerlemesini ve Performansını İzleme

Akıllı iş akışlarının sorunsuz yürütülmesini sağlamak için, her iş akışı bileşeninin ilerlemesini ve performansını izlemek önemlidir. Bu, iş akışı yaşam döngüsü boyunca temel metriklerin ve olayların takibini içerir.

İzlenmesi gereken önemli yönler şunlardır:

1. **İş Akışı Yürütme Süresi**: Her iş akışı bileşeninin görevini tamamlaması için gereken süreyi ölçün. Bu, performans darboğazlarını belirlemeye ve genel iş akışı verimliliğini optimize etmeye yardımcı olur.

2. **Kaynak Kullanımı**: Her iş akışı bileşeni tarafından CPU, bellek ve depolama gibi sistem kaynaklarının kullanımını izleyin. Bu, sistemin kapasitesi dahilinde çalıştığından ve iş yükünü etkili bir şekilde yönetebileceğinden emin olmanıza yardımcı olur.

3. **Hata Oranları ve İstisnalar**: İş akışı bileşenleri içindeki hataların ve istisnaların oluşumunu takip edin. Bu, potansiyel sorunları belirlemeye yardımcı olur ve proaktif hata yönetimi ve kurtarmayı mümkün kılar.

4. **Karar Noktaları ve Sonuçları**: İş akışı içindeki karar noktalarını ve yapay zeka destekli kararların sonuçlarını izleyin. Bu, yapay zeka bileşenlerinin davranışı ve etkinliği hakkında içgörü sağlar.

İzleme süreçleri tarafından toplanan veriler, gösterge panellerinde görüntülenebilir veya sistemin sağlığı hakkında sistem yöneticilerini bilgilendiren planlı raporlara girdi olarak kullanılabilir.

İzleme verileri, inceleme ve potansiyel eylem için yapay zeka destekli bir sistem yöneticisi sürecine beslenebilir!

Önemli Olayları ve Kararları Günlüğe Kaydetme

Günlük tutma, iş akışı yürütmesi sırasında meydana gelen önemli olaylar, kararlar ve istisnalar hakkındaki ilgili bilgileri yakalama ve depolama işlemini içeren temel bir uygulamadır.

Günlüğe kaydedilmesi gereken önemli yönler şunlardır:

1. **İş Akışı Başlatma ve Tamamlama**: Her iş akışı örneğinin başlangıç ve bitiş zamanlarını, giriş verileri ve kullanıcı bağlamı gibi ilgili meta verilerle birlikte kaydedin.

2. **Bileşen Yürütme**: Her iş akışı bileşeninin yürütme ayrıntılarını, giriş parametrelerini, çıkış sonuçlarını ve oluşturulan ara verileri içerecek şekilde kaydedin.

3. **Yapay Zeka Kararları ve Gerekçelendirme**: Yapay zeka bileşenlerinin verdiği kararları, temel gerekçelendirme veya güven skorlarıyla birlikte kaydedin. Bu, şeffaflık sağlar ve yapay zeka destekli kararların denetlenmesini mümkün kılar.

4. **İstisnalar ve Hata Mesajları**: İş akışı yürütmesi sırasında karşılaşılan istisnaları veya hata mesajlarını, yığın izi ve ilgili bağlam bilgileriyle birlikte kaydedin.

Günlük tutma, günlük dosyalarına yazma, günlükleri bir veritabanında saklama veya günlükleri merkezi bir günlük tutma hizmetine gönderme gibi çeşitli tekniklerle uygulanabilir. Esneklik, ölçeklenebilirlik ve uygulamanın mimarisiyle kolay entegrasyon sağlayan bir günlük tutma çerçevesi seçmek önemlidir.

İşte Ruby on Rails uygulamasında `ActiveSupport::Logger` sınıfı kullanılarak günlük tutmanın nasıl uygulanabileceğine dair bir örnek:

```
class WorkflowLogger
  def self.log(message, severity = :info)
    @logger ||= ActiveSupport::Logger.new('workflow.log')
    @logger.formatter ||= proc do |severity, datetime, progname, msg|
      "#{datetime} [#{severity}] #{msg}\n"
    end
    @logger.send(severity, message)
  end
end

# Usage example
WorkflowLogger.log("Workflow initiated for order ##{@order.id}")
WorkflowLogger.log("Payment processing completed successfully")
WorkflowLogger.log("Inventory check failed for item ##{item.id}", :error)
```

İş akışı bileşenleri ve yapay zeka karar noktaları boyunca günlük kayıt ifadelerini stratejik olarak yerleştirerek, geliştiriciler hata ayıklama, denetim ve analiz için değerli bilgiler elde edebilirler.

İzleme ve Günlük Tutmanın Faydaları

Akıllı iş akışı orkestrasyonunda izleme ve günlük tutmayı uygulamanın birçok faydası vardır:

1. **Hata Ayıklama ve Sorun Giderme**: Detaylı günlükler ve izleme verileri, geliştiricilerin sorunları hızlı bir şekilde tanımlamasına ve teşhis etmesine yardımcı

olur. İş akışı yürütme akışı, bileşen etkileşimleri ve karşılaşılan hatalar veya istisnalar hakkında içgörüler sağlar.

2. **Performans Optimizasyonu**: Performans metriklerinin izlenmesi, geliştiricilerin darboğazları belirlemesine ve iş akışı bileşenlerini daha iyi verimlilik için optimize etmesine olanak tanır. Yürütme süreleri, kaynak kullanımı ve diğer metrikleri analiz ederek, geliştiriciler sistemin genel performansını iyileştirmek için bilinçli kararlar alabilirler.

3. **Denetim ve Uyumluluk**: Önemli olayların ve kararların kaydedilmesi, düzenleyici uyumluluk ve hesap verebilirlik için bir denetim izi sağlar. Kuruluşların yapay zeka bileşenleri tarafından alınan eylemleri izlemesine ve iş kurallarına ve yasal gerekliliklere uygunluğunu sağlamasına olanak tanır.

4. **Sürekli İyileştirme**: İzleme ve günlük tutma verileri, akıllı iş akışlarının sürekli iyileştirilmesi için değerli girdiler olarak hizmet eder. Geçmiş verileri analiz ederek, kalıpları belirleyerek ve yapay zeka kararlarının etkinliğini ölçerek, geliştiriciler iş akışı orkestrasyon mantığını yinelemeli olarak iyileştirebilir ve geliştirebilir.

Önemli Hususlar ve En İyi Uygulamalar

Akıllı iş akışı orkestrasyonunda izleme ve günlük tutmayı uygularken, aşağıdaki en iyi uygulamaları göz önünde bulundurun:

1. **Net İzleme Metrikleri Tanımlayın**: İş akışının özel gereksinimlerine göre izlenmesi gereken temel metrikleri ve olayları belirleyin. Sistemin performansı, sağlığı ve davranışı hakkında anlamlı içgörüler sağlayan metriklere odaklanın.

2. **Ayrıntılı Günlük Tutmayı Uygulayın**: Günlük kayıt ifadelerinin iş akışı bileşenleri ve yapay zeka karar noktaları içinde uygun noktalara yerleştirildiğinden emin olun. Giriş parametreleri, çıkış sonuçları ve oluşturulan ara veriler gibi ilgili bağlam bilgilerini yakalayın.

3. **Yapılandırılmış Günlük Tutmayı Kullanın**: Günlük verilerinin kolay ayrıştırılması ve analizi için yapılandırılmış bir günlük tutma formatı benimseyin. Yapılandırılmış günlük tutma, günlük girişlerinin daha iyi aranabilirliğini, filtrelenmesini ve toplanmasını sağlar.

4. **Günlük Saklama ve Döndürmeyi Yönetin**: Günlük dosyalarının depolanmasını ve yaşam döngüsünü yönetmek için günlük saklama ve döndürme politikaları uygulayın. Yasal gereklilikler, depolama kısıtlamaları ve analiz ihtiyaçlarına göre uygun saklama süresini belirleyin. Mümkünse, günlük tutmayı Papertrail gibi üçüncü taraf bir hizmete aktarın.

5. **Hassas Bilgileri Güvence Altına Alın**: Kişisel olarak tanımlanabilir bilgiler (PII) veya gizli iş verileri gibi hassas bilgileri kaydederken dikkatli olun. Günlük dosyalarındaki hassas bilgileri korumak için veri maskeleme veya şifreleme gibi uygun güvenlik önlemlerini uygulayın.

6. **İzleme ve Uyarı Araçlarıyla Entegre Edin**: İzleme ve günlük tutma verilerinin toplanması, analizi ve görselleştirilmesi için izleme ve uyarı araçlarından yararlanın. Bu araçlar gerçek zamanlı içgörüler sağlayabilir, önceden tanımlanmış eşiklere dayalı uyarılar oluşturabilir ve proaktif sorun tespiti ve çözümünü kolaylaştırabilir. Bu araçlar arasında en sevdiğim Datadog'dur.

Kapsamlı izleme ve günlük tutma mekanizmalarını uygulayarak, geliştiriciler akıllı iş akışlarının davranışı ve performansı hakkında değerli içgörüler elde edebilirler. Bu içgörüler, yapay zeka destekli iş akışı orkestrasyon sistemlerinin etkili bir şekilde hata ayıklanmasını, optimizasyonunu ve sürekli iyileştirilmesini sağlar.

Ölçeklenebilirlik ve Performans Hususları

Ölçeklenebilirlik ve performans, akıllı iş akışı orkestrasyon sistemlerini tasarlarken ve uygularken dikkate alınması gereken kritik yönlerdir. Eşzamanlı iş akışlarının hacmi ve yapay zeka destekli bileşenlerin karmaşıklığı arttıkça, sistemin iş yükünü

verimli bir şekilde yönetebilmesi ve artan talepleri karşılamak için sorunsuz bir şekilde ölçeklenmesi önemli hale gelir.

Yüksek Hacimli Eşzamanlı İş Akışlarını Yönetme

Akıllı iş akışı orkestrasyon sistemlerinin genellikle çok sayıda eşzamanlı iş akışını yönetmesi gerekir. Ölçeklenebilirliği sağlamak için aşağıdaki stratejileri göz önünde bulundurun:

1. **Asenkron İşleme**: İş akışı bileşenlerinin yürütülmesini ayırmak için asenkron işleme mekanizmaları uygulayın. Bu, sistemin her bileşenin tamamlanmasını beklemeden veya engellemeden birden çok iş akışını eşzamanlı olarak yönetmesine olanak tanır. Asenkron işleme, mesaj kuyrukları, olay odaklı mimariler veya Sidekiq gibi arka plan iş işleme çerçeveleri kullanılarak gerçekleştirilebilir.

2. **Dağıtık Mimari**: Sistem mimarisini AWS Lambda gibi sunucusuz bileşenleri kullanacak şekilde veya ana uygulama sunucunuzun yanında iş yükünü birden çok düğüm veya sunucu arasında dağıtacak şekilde tasarlayın. Bu, artan iş akışı hacimlerini yönetmek için ek düğümlerin eklenebileceği yatay ölçeklenebilirliği sağlar.

3. **Paralel Yürütme**: İş akışları içinde paralel yürütme fırsatlarını belirleyin. Bazı iş akışı bileşenleri birbirinden bağımsız olabilir ve eşzamanlı olarak yürütülebilir. Çoklu iş parçacığı veya dağıtık görev kuyrukları gibi paralel işleme tekniklerinden yararlanarak, sistem kaynak kullanımını optimize edebilir ve genel iş akışı yürütme süresini azaltabilir.

Yapay Zeka Destekli Bileşenlerin Performansını Optimize Etme

Makine öğrenimi modelleri veya doğal dil işleme motorları gibi yapay zeka destekli bileşenler, hesaplama açısından yoğun olabilir ve iş akışı düzenleme sisteminin genel performansını etkileyebilir. Yapay zeka bileşenlerinin performansını optimize etmek için aşağıdaki teknikleri göz önünde bulundurun:

1. **Önbellekleme**: Eğer yapay zeka işleminiz tamamen üretimsel ise ve sohbet tamamlamalarını oluşturmak için gerçek zamanlı bilgi araması veya harici entegrasyonlar içermiyorsa, sık erişilen veya hesaplama açısından maliyetli işlemlerin sonuçlarını depolamak ve yeniden kullanmak için önbellekleme mekanizmalarını araştırabilirsiniz.

2. **Model Optimizasyonu**: İş akışı bileşenlerinde yapay zeka modellerini kullanma şeklinizi sürekli olarak optimize edin. Bu, *İstem Damıtma* gibi teknikleri içerebilir veya basitçe yeni modeller kullanıma sunuldukça bunları test etmek olabilir.

3. **Toplu İşleme**: GPT-4 sınıfı modellerle çalışıyorsanız, birden fazla veri noktasını veya isteği tek tek işlemek yerine tek bir toplu işlemde işlemek için toplu işleme tekniklerinden yararlanabilirsiniz. Verileri toplu olarak işleyerek, sistem kaynak kullanımını optimize edebilir ve tekrarlanan model isteklerinin ek yükünü azaltabilir.

Performans İzleme ve Profil Çıkarma

Akıllı iş akışı düzenleme sisteminin performans darboğazlarını belirlemek ve ölçeklenebilirliğini optimize etmek için izleme ve profil çıkarma mekanizmalarının uygulanması çok önemlidir. Aşağıdaki yaklaşımları göz önünde bulundurun:

1. **Performans Metrikleri**: Yanıt süresi, verim, kaynak kullanımı ve gecikme süresi gibi temel performans metriklerini tanımlayın ve takip edin. Bu metrikler, sistemin performansı hakkında içgörüler sağlar ve optimizasyon için alanları belirlemeye yardımcı olur. Popüler yapay zeka model toplayıcısı OpenRouter, her API yanıtında Host[1] ve Hız[2] metriklerini içererek bu temel metrikleri takip etmeyi kolaylaştırır.

2. **Profil Çıkarma Araçları**: Bireysel iş akışı bileşenlerinin ve yapay zeka işlemlerinin performansını analiz etmek için profil çıkarma araçlarını kullanın. Profil çıkarma araçları, performans açısından kritik noktaları, verimsiz kod yollarını veya kaynak

[1]Host, model sunucusundan akan üretimin ilk baytını almak için geçen süredir, diğer bir deyişle "ilk bayta kadar geçen süre."

[2]Hız, tamamlama belirteçlerinin sayısının toplam üretim süresine bölünmesiyle hesaplanır. Akışlı olmayan istekler için gecikme süresi üretim süresinin bir parçası olarak kabul edilir.

yoğun işlemleri belirlemeye yardımcı olabilir. Popüler profil çıkarma araçları arasında New Relic, Scout veya programlama dili veya çerçeve tarafından sağlanan yerleşik profil çıkarıcılar bulunur.

3. **Yük Testi**: Sistemin farklı eşzamanlı iş yükü seviyelerindeki performansını değerlendirmek için yük testi yapın. Yük testi, sistemin ölçeklenebilirlik sınırlarını belirlemeye, performans düşüşlerini tespit etmeye ve sistemin performanstan ödün vermeden beklenen trafiği karşılayabileceğinden emin olmaya yardımcı olur.

4. **Sürekli İzleme**: Performans sorunlarını ve darboğazları proaktif olarak tespit etmek için sürekli izleme ve uyarı mekanizmaları uygulayın. Temel performans göstergelerini (KPI'lar) takip etmek ve önceden tanımlanmış eşikler aşıldığında bildirim almak için izleme panoları ve uyarılar kurun. Bu, performans sorunlarının hızlı bir şekilde tanımlanmasını ve çözülmesini sağlar.

Ölçeklendirme Stratejileri

Akıllı iş akışı düzenleme sisteminin artan iş yüklerini karşılaması ve ölçeklenebilirliğini sağlaması için aşağıdaki ölçeklendirme stratejilerini göz önünde bulundurun:

1. **Dikey Ölçeklendirme**: Dikey ölçeklendirme, daha yüksek iş yüklerini karşılamak için bireysel düğümlerin veya sunucuların kaynaklarını (örneğin, CPU, bellek) artırmayı içerir. Bu yaklaşım, sistemin karmaşık iş akışlarını veya yapay zeka işlemlerini yönetmek için daha fazla işlem gücü veya belleğe ihtiyaç duyduğu durumlarda uygundur.

2. **Yatay Ölçeklendirme**: Yatay ölçeklendirme, iş yükünü dağıtmak için sisteme daha fazla düğüm veya sunucu eklemeyi içerir. Bu yaklaşım, sistemin çok sayıda eşzamanlı iş akışını yönetmesi gerektiğinde veya iş yükü birden fazla düğüm arasında kolayca dağıtılabildiğinde etkilidir. Yatay ölçeklendirme, trafiğin eşit dağılımını sağlamak için dağıtık bir mimari ve yük dengeleme mekanizmaları gerektirir.

3. **Otomatik Ölçeklendirme**: İş yükü talebine göre düğüm veya kaynak sayısını otomatik olarak ayarlayan otomatik ölçeklendirme mekanizmalarını uygulayın.

Otomatik ölçeklendirme, sistemin gelen trafiğe bağlı olarak dinamik olarak büyümesine veya küçülmesine olanak tanıyarak optimal kaynak kullanımı ve maliyet verimliliği sağlar. Amazon Web Services (AWS) veya Google Cloud Platform (GCP) gibi bulut platformları, akıllı iş akışı düzenleme sistemleri için kullanılabilecek otomatik ölçeklendirme yetenekleri sunar.

Performans Optimizasyon Teknikleri

Ölçeklendirme stratejilerine ek olarak, akıllı iş akışı düzenleme sisteminin verimliliğini artırmak için aşağıdaki performans optimizasyon tekniklerini göz önünde bulundurun:

1. **Verimli Veri Depolama ve Erişim**: İş akışı bileşenleri tarafından kullanılan veri depolama ve erişim mekanizmalarını optimize edin. Veri yoğun işlemlerin gecikme süresini en aza indirmek ve performansını iyileştirmek için verimli veritabanı indeksleme, sorgu optimizasyon teknikleri ve veri önbellekleme kullanın.

2. **Eşzamansız G/Ç**: Sistemin yanıt verebilirliğini artırmak ve bloklamayı önlemek için eşzamansız G/Ç işlemlerinden yararlanın. Eşzamansız G/Ç, sistemin G/Ç işlemlerinin tamamlanmasını beklemeden birden çok isteği eşzamanlı olarak işlemesine olanak tanıyarak kaynak kullanımını en üst düzeye çıkarır.

3. **Verimli Serileştirme ve Ters Serileştirme**: İş akışı bileşenleri arasındaki veri alışverişi için kullanılan serileştirme ve ters serileştirme süreçlerini optimize edin. Veri serileştirme yükünü azaltmak ve bileşenler arası iletişimin performansını artırmak için Protocol Buffers veya MessagePack gibi verimli serileştirme formatlarını kullanın.

Ruby tabanlı uygulamalar için Universal ID kullanmayı düşünün. Universal ID, hem MessagePack hem de Brotli'den yararlanır (hız ve en iyi sınıf veri sıkıştırma için oluşturulmuş bir kombinasyon). Bu kütüphaneler bir araya getirildiğinde, Protocol Buffers'a kıyasla %30'a kadar daha hızlı ve %2-5 arasında sıkıştırma oranlarına sahiptir.

4. **Sıkıştırma ve Kodlama**: İş akışı bileşenleri arasında aktarılan verinin boyutunu azaltmak için sıkıştırma ve kodlama tekniklerini uygulayın. gzip veya Brotli gibi sıkıştırma algoritmaları, ağ bant genişliği kullanımını önemli ölçüde azaltabilir ve sistemin genel performansını iyileştirebilir.

Akıllı iş akışı orkestrasyon sistemlerinin tasarımı ve uygulaması sırasında ölçeklenebilirlik ve performans yönlerini göz önünde bulundurarak, sisteminizin yüksek hacimli eşzamanlı iş akışlarını yönetebilmesini, yapay zeka destekli bileşenlerin performansını optimize edebilmesini ve artan talepleri karşılamak için sorunsuz bir şekilde ölçeklenmesini sağlayabilirsiniz. İş yükü ve karmaşıklık zaman içinde arttıkça sistemin performansını ve yanıt verebilirliğini korumak için sürekli izleme, profil çıkarma ve optimizasyon çalışmaları gereklidir.

İş Akışlarının Test Edilmesi ve Doğrulanması

Test etme ve doğrulama, akıllı iş akışı orkestrasyon sistemlerinin geliştirilmesi ve bakımının kritik yönleridir. Yapay zeka destekli iş akışlarının karmaşık doğası göz önüne alındığında, her bileşenin beklendiği gibi çalıştığından, genel iş akışının doğru şekilde davrandığından ve yapay zeka kararlarının doğru ve güvenilir olduğundan emin olmak esastır. Bu bölümde, akıllı iş akışlarını test etme ve doğrulama için çeşitli teknikleri ve hususları inceleyeceğiz.

İş Akışı Bileşenlerinin Birim Testi

Birim testi, doğruluklarını ve sağlamlıklarını doğrulamak için tek tek iş akışı bileşenlerinin test edilmesini içerir. Yapay zeka destekli iş akışı bileşenlerinin birim testini yaparken şunları göz önünde bulundurun:

1. **Girdi Doğrulama**: Bileşenin geçerli ve geçersiz veriler dahil olmak üzere farklı türdeki girdileri işleme yeteneğini test edin. Bileşenin uç durumları düzgün bir şekilde ele aldığını ve uygun hata mesajları veya istisnalar sağladığını doğrulayın.

2. **Çıktı Doğrulama**: Bileşenin belirli bir girdi seti için beklenen çıktıyı ürettiğini doğrulayın. Doğruluğu sağlamak için gerçek çıktıyı beklenen sonuçlarla karşılaştırın.

3. **Hata Yönetimi**: Geçersiz girdi, kaynak kullanılamaması veya beklenmeyen istisnalar gibi çeşitli hata senaryolarını simüle ederek bileşenin hata yönetimi mekanizmalarını test edin. Bileşenin hataları uygun şekilde yakaladığını ve ele aldığını doğrulayın.

4. **Sınır Koşulları**: Boş girdi, maksimum girdi boyutu veya aşırı değerler gibi sınır koşulları altında bileşenin davranışını test edin. Bileşenin çökmeden veya yanlış sonuçlar üretmeden bu koşulları düzgün bir şekilde ele aldığından emin olun.

İşte RSpec test çatısı kullanılarak Ruby'de bir iş akışı bileşeni için birim testi örneği:

```
RSpec.describe OrderValidator do
  describe '#validate' do
    context 'when order is valid' do
      let(:order) { build(:order) }

      it 'returns true' do
        expect(subject.validate(order)).to be true
      end
    end

    context 'when order is invalid' do
      let(:order) { build(:order, total_amount: -100) }

      it 'returns false' do
        expect(subject.validate(order)).to be false
      end
    end
  end
end
```

Bu örnekte, `OrderValidator` bileşeni iki test durumu kullanılarak test edilmektedir: biri geçerli bir sipariş için, diğeri geçersiz bir sipariş için. Test durumları, `validate` metodunun siparişin geçerliliğine bağlı olarak beklenen boolean değerini döndürdüğünü doğrular.

İş Akışı Etkileşimlerinin Entegrasyon Testi

Entegrasyon testi, farklı iş akışı bileşenleri arasındaki etkileşimleri ve veri akışını doğrulamaya odaklanır. Bileşenlerin sorunsuz bir şekilde birlikte çalıştığını ve beklenen sonuçları ürettiğini garanti eder. Akıllı iş akışlarının entegrasyon testini yaparken şunları göz önünde bulundurun:

1. **Bileşen Etkileşimi**: İş akışı bileşenleri arasındaki iletişimi ve veri alışverişini test edin. Bir bileşenin çıktısının, iş akışındaki bir sonraki bileşene doğru şekilde girdi olarak aktarıldığını doğrulayın.

2. **Veri Tutarlılığı**: Verilerin iş akışı boyunca tutarlı ve doğru kaldığından emin olun. Veri dönüşümlerinin, hesaplamaların ve birleştirmelerin doğru şekilde gerçekleştirildiğini doğrulayın.

3. **İstisna Yayılımı**: İstisnaların ve hataların iş akışı bileşenleri arasında nasıl yayıldığını ve ele alındığını test edin. İstisnaların yakalandığını, kaydedildiğini ve iş akışı kesintisini önlemek için uygun şekilde ele alındığını doğrulayın.

4. **Eşzamansız Davranış**: Eğer iş akışı eşzamansız bileşenler veya paralel yürütme içeriyorsa, koordinasyon ve senkronizasyon mekanizmalarını test edin. İş akışının eşzamanlı ve eşzamansız senaryolarda doğru şekilde davrandığından emin olun.

İşte Ruby'de RSpec test çerçevesi kullanarak bir iş akışı için entegrasyon testi örneği:

```ruby
RSpec.describe OrderProcessingWorkflow do

  let(:order) { build(:order) }

  it 'processes the order successfully' do
    expect(OrderValidator).to receive(:validate).and_return(true)
    expect(InventoryManager).to receive(:check_availability).and_return(true)
    expect(PaymentProcessor).to receive(:process_payment).and_return(true)
    expect(ShippingService).to receive(:schedule_shipping).and_return(true)

    workflow = OrderProcessingWorkflow.new(order)
    result = workflow.process

    expect(result).to be true
    expect(order.status).to eq('processed')
  end

end
```

Bu örnekte, `OrderProcessingWorkflow` farklı iş akışı bileşenleri arasındaki etkileşimlerin doğrulanmasıyla test edilmektedir. Test senaryosu, her bileşenin davranışı için beklentiler oluşturur ve iş akışının siparişi başarıyla işleyerek sipariş durumunu uygun şekilde güncellediğinden emin olur.

Yapay Zeka Karar Noktalarının Test Edilmesi

Yapay zeka karar noktalarını test etmek, yapay zeka destekli iş akışlarının doğruluğunu ve güvenilirliğini sağlamak için çok önemlidir. Yapay zeka karar noktalarını test ederken şunları göz önünde bulundurun:

1. **Karar Doğruluğu**: Yapay zeka bileşeninin giriş verileri ve eğitilmiş model temelinde doğru kararlar verdiğini doğrulayın. Yapay zeka kararlarını beklenen sonuçlar veya temel gerçeklik verileriyle karşılaştırın.

2. **Uç Durumlar**: Yapay zeka bileşeninin uç durumlardaki ve olağandışı senaryolardaki davranışını test edin. Yapay zeka bileşeninin bu durumları düzgün bir şekilde ele aldığından ve makul kararlar verdiğinden emin olun.

3. **Yanlılık ve Adillik**: Yapay zeka bileşenini olası yanlılıklar açısından değerlendirin ve adil, tarafsız kararlar verdiğinden emin olun. Bileşeni çeşitli giriş verileriyle test edin ve sonuçları ayrımcı örüntüler açısından analiz edin.

4. **Açıklanabilirlik**: Eğer yapay zeka bileşeni kararları için açıklamalar veya gerekçeler sunuyorsa, açıklamaların doğruluğunu ve netliğini doğrulayın. Açıklamaların temel karar verme süreciyle uyumlu olduğundan emin olun.

İşte Ruby'de RSpec test çerçevesi kullanılarak yapay zeka karar noktasının test edilmesine bir örnek:

```
RSpec.describe FraudDetector do
  describe '#detect_fraud' do
    context 'when transaction is fraudulent' do
      let(:tx) do
        build(:transaction, amount: 10_000, location: 'High-Risk Country')
      end

      it 'returns true' do
        expect(subject.detect_fraud(tx)).to be true
      end
    end

    context 'when transaction is legitimate' do
      let(:tx) do
        build(:transaction, amount: 100, location: 'Low-Risk Country')
      end

      it 'returns false' do
        expect(subject.detect_fraud(tx)).to be false
      end
    end
  end
end
```

Bu örnekte, `FraudDetector` yapay zeka bileşeni iki test vakasıyla test edilmektedir: biri sahte işlem için, diğeri meşru işlem için. Test vakaları, `detect_fraud` metodunun işlemin özelliklerine bağlı olarak beklenen boolean değerini döndürdüğünü doğrular.

Uçtan Uca Test

Uçtan uca test, başlangıçtan sona kadar tüm iş akışının test edilmesini, gerçek dünya senaryolarının ve kullanıcı etkileşimlerinin simüle edilmesini içerir. İş akışının doğru şekilde davrandığından ve istenen sonuçları ürettiğinden emin olur. Akıllı iş akışları için uçtan uca test yaparken şunları göz önünde bulundurun:

1. **Kullanıcı Senaryoları**: Yaygın kullanıcı senaryolarını belirleyin ve iş akışının bu senaryolar altındaki davranışını test edin. İş akışının kullanıcı girdilerini doğru şekilde işlediğini, uygun kararlar aldığını ve beklenen çıktıları ürettiğini doğrulayın.

2. **Veri Doğrulama**: İş akışının veri tutarsızlıklarını veya güvenlik açıklarını önlemek için kullanıcı girdilerini doğruladığından ve temizlediğinden emin olun. İş akışını geçerli ve geçersiz veriler dahil olmak üzere çeşitli giriş verileriyle test edin.

3. **Hata Kurtarma**: İş akışının hatalardan ve istisnalardan kurtulma yeteneğini test edin. Hata senaryolarını simüle edin ve iş akışının bunları düzgün bir şekilde ele aldığını, hataları kaydettiğini ve uygun kurtarma işlemlerini gerçekleştirdiğini doğrulayın.

4. **Performans ve Ölçeklenebilirlik**: İş akışının farklı yük koşulları altındaki performansını ve ölçeklenebilirliğini değerlendirin. İş akışını çok sayıda eşzamanlı istekle test edin ve yanıt sürelerini, kaynak kullanımını ve genel sistem kararlılığını ölçün.

İşte Ruby dilinde RSpec test çerçevesi ve kullanıcı etkileşimlerini simüle etmek için Capybara kütüphanesi kullanılarak yazılmış bir iş akışı için uçtan uca test örneği:

```
RSpec.describe 'Order Processing Workflow' do
  scenario 'User places an order successfully' do
    visit '/orders/new'
    fill_in 'Product', with: 'Sample Product'
    fill_in 'Quantity', with: '2'
    fill_in 'Shipping Address', with: '123 Main St'
    click_button 'Place Order'

    expect(page).to have_content('Order Placed Successfully')
    expect(Order.count).to eq(1)
    expect(Order.last.status).to eq('processed')
  end
end
```

Bu örnekte, uçtan uca test, bir kullanıcının web arayüzü üzerinden sipariş vermesini simüle eder. Gerekli form alanlarını doldurur, siparişi gönderir ve siparişin başarıyla işlendiğini, uygun onay mesajını gösterdiğini ve sipariş durumunun veritabanında güncellendiğini doğrular.

Sürekli Entegrasyon ve Dağıtım

Akıllı iş akışlarının güvenilirliğini ve bakım yapılabilirliğini sağlamak için, test etme ve doğrulama işlemlerinin sürekli entegrasyon ve dağıtım (CI/CD) pipeline'ına entegre edilmesi önerilir. Bu, iş akışı değişikliklerinin üretime dağıtılmadan önce otomatik olarak test edilmesine ve doğrulanmasına olanak tanır. Aşağıdaki uygulamaları göz önünde bulundurun:

1. **Otomatik Test Yürütme**: İş akışı kod tabanında değişiklik yapıldığında test paketinin otomatik olarak çalıştırılması için CI/CD pipeline'ını yapılandırın. Bu, herhangi bir regresyonun veya başarısızlığın geliştirme sürecinin erken aşamalarında tespit edilmesini sağlar.

2. **Test Kapsamı İzleme**: İş akışı bileşenlerinin ve yapay zeka karar noktalarının test kapsamını ölçün ve izleyin. Kritik yolların ve senaryoların kapsamlı bir şekilde test edilmesini sağlamak için yüksek test kapsamını hedefleyin.

3. **Sürekli Geri Bildirim**: Test sonuçlarını ve kod kalitesi metriklerini geliştirme iş akışına entegre edin. Geliştiricilere testlerin durumu, kod kalitesi ve CI/CD süreci sırasında tespit edilen sorunlar hakkında sürekli geri bildirim sağlayın.

4. **Hazırlık Ortamları**: İş akışını üretim ortamını yakından yansıtan hazırlık ortamlarına dağıtın. Altyapı, yapılandırma veya veri entegrasyonuyla ilgili sorunları yakalamak için hazırlık ortamında ek test ve doğrulama gerçekleştirin.

5. **Geri Alma Mekanizmaları**: Dağıtım başarısızlıkları veya üretimde tespit edilen kritik sorunlar için geri alma mekanizmaları uygulayın. Kesinti süresini ve kullanıcılar üzerindeki etkiyi en aza indirmek için iş akışının hızlı bir şekilde önceki kararlı sürüme geri döndürülebilmesini sağlayın.

Akıllı iş akışlarının geliştirme yaşam döngüsü boyunca test ve doğrulama işlemlerini dahil ederek, organizasyonlar yapay zeka destekli sistemlerinin güvenilirliğini, doğruluğunu ve bakım yapılabilirliğini sağlayabilirler. Düzenli test ve doğrulama, hataların yakalanmasına, regresyonların önlenmesine ve iş akışının davranışına ve sonuçlarına olan güvenin artmasına yardımcı olur.

Kısım 2: Desenler

Bildirim Mühendisliği

Düşünce Zinciri

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnekler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

İçerik Üretimi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Yapılandırılmış Varlık Oluşturma

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

LLM Ajan Yönlendirmesi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Faydalar ve Dikkat Edilecek Noktalar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Mod Değişimi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ne Zaman Kullanılmalı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Rol Atama

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ne Zaman Kullanılır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnekler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Prompt Object

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

İstem Şablonu

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Faydaları ve Dikkat Edilmesi Gerekenler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ne Zaman Kullanılmalı:

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Yapılandırılmış Girdi/Çıktı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Yapılandırılmış Giriş/Çıkışın Ölçeklendirilmesi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Faydalar ve Dikkat Edilecek Noktalar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Prompt Zincirleme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ne Zaman Kullanılmalı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek: Olympia'nın Kullanıcı Dahil Etme Süreci

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Prompt Yeniden Yazıcı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Response Fencing

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Faydalar ve Dikkat Edilecek Noktalar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Hata İşleme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Sorgu Çözümleyici

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uygulama

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Sözcük Türü (POS) Etiketleme ve Varlık İsmi Tanıma (NER)

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Niyet Sınıflandırma

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Anahtar Kelime Çıkarımı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Faydalar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Sorgu Yeniden Yazıcı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Faydaları

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ventriloquist

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ne Zaman Kullanmalı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ayrık Bileşenler

Yüklem

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ne Zaman Kullanılmalı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

API Facade

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Temel Faydaları

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ne Zaman Kullanmalı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Kimlik Doğrulama ve Yetkilendirme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

İstek İşleme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Yanıt Biçimlendirme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Hata Yönetimi ve Uç Durumlar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ölçeklenebilirlik ve Performans Değerlendirmeleri

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Diğer Tasarım Desenleriyle Karşılaştırma

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Sonuç Yorumlayıcı

Nasıl Çalışır

Ne Zaman Kullanılmalı

Örnek

Sanal Makine

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Ne Zaman Kullanılmalı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Büyünün Ardında

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Spesifikasyon ve Test

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Davranışın Belirlenmesi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Test Senaryolarının Yazılması

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek: Çevirmen Bileşeninin Test Edilmesi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

HTTP Etkileşimlerinin Tekrarı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

İnsan Kontrolünde (HITL)

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Üst Düzey Kalıplar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Hibrit Zeka

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uyarlanabilir Yanıt

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

İnsan-YZ Rol Değişimi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Yükseltme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Temel Faydalar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Gerçek Dünya Uygulaması: Sağlık Hizmetleri

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Geri Bildirim Döngüsü

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uygulamalar ve Örnekler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

İnsan Geri Bildirim Entegrasyonunda İleri Teknikler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Pasif Bilgi Yayılımı

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bağlamsal Bilgi Görüntüleme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Proaktif Bildirimler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Açıklayıcı İçgörüler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Etkileşimli Keşif

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Temel Faydalar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uygulamalar ve Örnekler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

İşbirlikçi Karar Verme (CDM)

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Sürekli Öğrenme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uygulamalar ve Örnekler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Etik Hususlar

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

İnsan Destekli Sistemlerin AI Risklerini Azaltmadaki Rolü

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Teknolojik İlerlemeler ve Gelecek Görünümü

İME Sistemlerinin Zorlukları ve Sınırlamaları

Akıllı Hata Yönetimi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Geleneksel Hata Yönetimi Yaklaşımları

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bağlamsal Hata Teşhisi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bağlamsal Hata Teşhisi için Bildirim Mühendisliği

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Bağlamsal Hata Teşhisi için Geri Getirme Destekli Üretim

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Akıllı Hata Raporlama

Öngörücü Hata Önleme

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Akıllı Hata Kurtarma

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Kişiselleştirilmiş Hata İletişimi

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Uyarlanabilir Hata İşleme İş Akışı

Nasıl Çalışır

Kalite Kontrol

Eval

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Problem

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Çözüm

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Değerlendirmeler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Altın Referansları Anlamak

Referanssız Değerlendirmeler Nasıl Çalışır

Koruma Mekanizması

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Problem

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Çözüm

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Nasıl Çalışır

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Örnek

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Dikkat Edilmesi Gerekenler

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Koruma Mekanizmaları ve Değerlendirmeler: Madalyonun İki Yüzü

Koruma Mekanizmaları ve Referanssız Değerlendirmelerin Birbirinin Yerine Kullanılabilirliği

Çift Amaçlı Koruma Mekanizmalarının ve Değerlendirmelerin Uygulanması

Sözlük

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Sözlük

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

A

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

B

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

C

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

D

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

E

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

F

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

G

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

H

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

I

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

J

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

K

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

L

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

M

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

N

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

O

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

P

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Q

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

R

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

S

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

T

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

U

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

V

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

W

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Z

Bu içerik örnek kitapta mevcut değildir. Kitabı Leanpub'tan satın almak için http://leanpub.com/patterns-of-application-development-using-ai-tr.

Index