# Prompt Mühendisliği

Yazar Lee Boonstra

Google

Çeviren Skymod Teknoloji

## Teşekkür

**İçeriğe katkıda bulunanlar**

Michael Sherman Yuan Cao

Erick Armbrust Anant Nawalgaria Antonio Gulli Simone Cammel

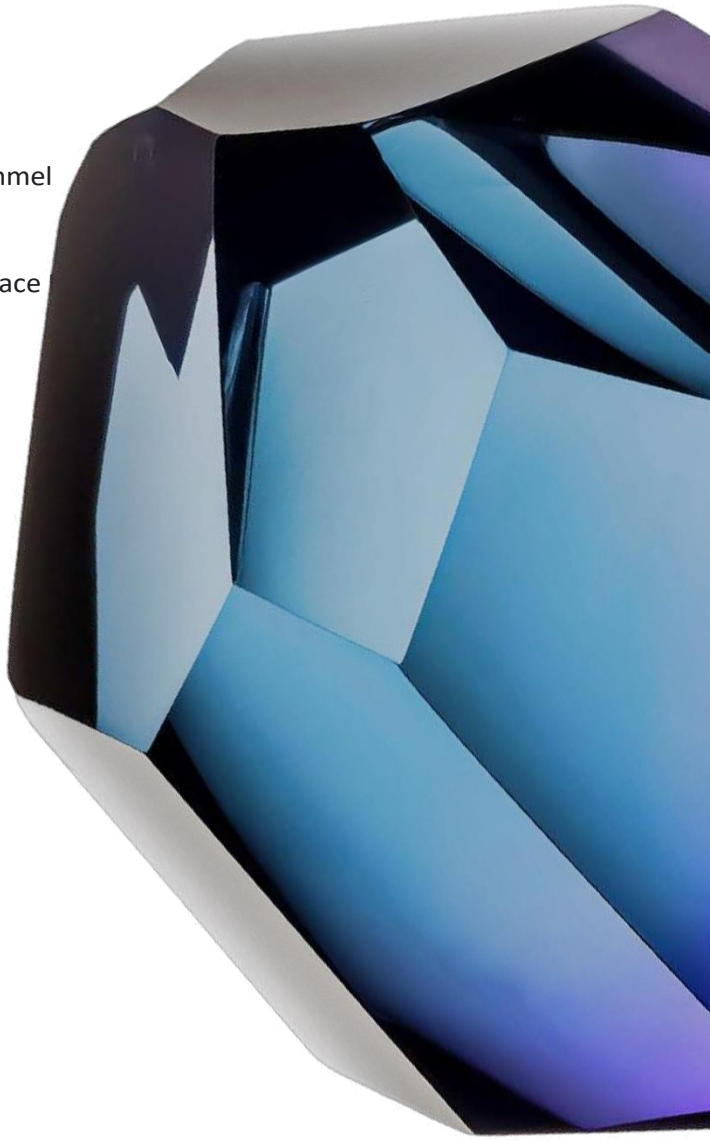
**Küratörler ve Editörler** Antonio Gulli Anant Nawalgaria Grace

**Teknik Yazar**

Joey Haymaker

**Tasarımcı**

Michael Lanning



Şubat 2025

# İçindekiler tablosu



> Veri bilimci veya makine öğrenimi mühendisi olmanıza gerek yok - herkes bir bilgi promptu yazabilir.

# Giriş

Büyük bir dil modeli girdisi ve çıktısı düşünüldüğünde, bir metin promptu (bazen görüntü promptları gibi diğer modaliteler de eşlik eder) belirli bir çıktıyı tahmin etmek için modelin kullandığı girdidir.
Bir veri bilimcisi veya makine öğrenimi mühendisi olmanıza gerek yok - herkes bir bilgi promptu yazabilir. Ancak en etkili promptu oluşturmak karmaşık olabilir. İpucunuzun birçok yönü etkinliğini etkiler: kullandığınız model, modelin eğitim verileri, model konfigürasyonları, kelime seçiminiz, stil ve ton, yapı ve bağlam önemlidir. Bu nedenle, prompt mühendisliği yinelemeli bir süreçtir. Yetersiz promptlar belirsiz, yanlış yanıtlara yol açabilir ve modelin anlamlı çıktı sağlama becerisini engelleyebilir.

Gemini chatbot ile sohbet ettiğinizde,[1]temel olarak promptlar yazarsınız, ancak bu teknik doküman Vertex AI içinde veya API kullanarak Gemini modeli için promptlar yazmaya odaklanmaktadır, çünkü modele doğrudan prompt vererek sıcaklık (temperature) vb. gibi yapılandırmaya erişebilirsiniz.

Bu teknik dokümanda prompt mühendisliği ayrıntılı olarak ele alınmaktadır. Hazırlıklı olmanıza yardımcı olacak çeşitli prompt tekniklerini inceleyecek ve bir prompt uzmanı olmanız için ipuçlarını ve en iyi uygulamaları paylaşacağız. Ayrıca, promptları hazırlarken karşılaşabileceğiniz bazı zorlukları da tartışacağız.

# Prompt Mühendisliği

LLM'nin nasıl çalıştığını hatırlayın; bu bir tahmin motorudur. Model sıralı metni girdi olarak alır ve ardından eğitildiği verilere dayanarak bir sonraki belirtecin ne olması gerektiğini tahmin eder. LLM, bunu tekrar tekrar yapmak ve bir sonraki belirteci tahmin etmek için daha önce tahmin edilen tokenı sıralı metnin sonuna eklemek üzere işlevselleştirilmiştir. Bir sonraki token tahmini, önceki tokenlarda ne olduğu ile LLM'nin eğitimi sırasında gördükleri arasındaki ilişkiye dayanır.

Bir prompt yazdığınızda, LLM'yi doğru token dizisini tahmin edecek şekilde ayarlamaya çalışırsınız. İstem mühendisliği, LLM'leri doğru çıktılar üretmeye yönlendiren yüksek kaliteli promptlar tasarlama sürecidir. Bu süreç, en iyi promptu bulmak için denemeler yapmak, prompt uzunluğunu optimize etmek ve görev için bir promptu yazma stilini ve yapısını aşağıdakilerle ilişkili olarak değerlendirmeyi içerir. Doğal dil işleme ve LLM'ler bağlamında, prompt, bir yanıt veya tahmin oluşturmak için modele sağlanan bir girdidir.

Bu promptlar, metin özetleme, bilgi çıkarma, soru ve cevaplama, metin sınıflandırma, dil veya kod çevirisi, kod oluşturma ve kod dokümantasyonu veya muhakemesi gibi çeşitli anlama ve oluşturma görevlerini gerçekleştirmek için kullanılabilir.

Lütfen basit ve etkili yönlendirme örnekleri içeren Google'ın yönlendirme kılavuzlarına[2,3]başvurmaktan çekinmeyin.

Komut promptu mühendisliği yaparken, bir model seçerek başlayabilirsiniz. Vertex AI, GPT, Claude, Gemini dil modellerini veya Gemma veya LLaMA gibi açık kaynaklı bir modeli kullanmanızdan bağımsız olarak, promptların dil modeliniz için optimize edilmesi gerekebilir.

Komut promptunun yanı sıra, bir büyük dil modelinin çeşitli yapılandırmalarıyla da uğraşmanız gerekecektir.

# Büyük dil modeli çıktı yapılandırması

Modelinizi seçtikten sonra model konfigürasyonunu belirlemeniz gerekecektir. Çoğu büyük dil modeli, modelin çıktısını kontrol eden çeşitli yapılandırma seçenekleriyle birlikte gelir. Etkili prompt mühendisliği, bu konfigürasyonların göreviniz için en uygun şekilde ayarlanmasını gerektirir.

## Çıktı uzunluğu

Önemli bir yapılandırma ayarı, bir yanıtta üretilecek token sayısıdır. Daha fazla token üretmek LLM'nin daha fazla hesaplama yapmasını gerektirir, bu da daha yüksek enerji tüketimine, potansiyel olarak daha yavaş yanıt sürelerine ve daha yüksek maliyetlere yol açar.

Büyük dil modelinin çıktı uzunluğunun azaltılması, modelin oluşturduğu çıktıda biçimsel veya metinsel olarak daha öz olmasına neden olmaz, sadece sınıra ulaşıldığında büyük dil modeli daha fazla token tahmin etmeyi bırakmasına neden olur. İhtiyaçlarınız daha kısa bir çıktı uzunluğu gerektiriyorsa, muhtemelen komut promptunuz da buna uyacak şekilde tasarlamanız gerekecektir.

Çıktı uzunluğu kısıtlaması, büyük dil modelinin istediğiniz yanıttan sonra gereksiz tokenlar yaymaya devam edeceği ReAct gibi bazı büyük dil modeli komut promptu teknikleri için özellikle önemlidir.

Unutmayın, daha fazla token üretmek büyük dil modelinin daha fazla hesaplama yapmasını gerektirir, bu da daha yüksek enerji tüketimine ve potansiyel olarak daha yavaş yanıt sürelerine yol açar, bu da daha yüksek maliyetlere neden olur.

## Örnekleme kontrolleri

Büyük dil modelleri resmi olarak tek bir tokenı tahmin etmez. Bunun yerine, bir sonraki tokenın ne olabileceğine dair olasılıkları tahmin eder ve kelime dağarcığındaki her token bir olasılık alır. Bu token olasılıkları daha sonra bir sonraki üretilen tokenın ne olacağını belirlemek için örneklenir.
Sıcaklık (Temperature), top-k ve top-p, tek bir çıktı tokenı seçmek için tahmin edilen token olasılıklarının nasıl işleneceğini belirleyen en yaygın yapılandırma ayarlarıdır.

### Sıcaklık (Temperature)

Sıcaklık (Temperature), token seçimindeki rastgelelik derecesini kontrol eder. Daha düşük sıcaklıklar daha deterministik bir yanıt bekleyen promptlar için iyiyken, daha yüksek sıcaklıklar daha çeşitli veya beklenmedik sonuçlara yol açabilir. 0'lık bir sıcaklık (Greedy decoding)

deterministik: en yüksek olasılıklı token her zaman seçilir (ancak iki token aynı en yüksek tahmin edilen olasılığa sahipse, eşitlik bozmanın nasıl uygulandığına bağlı olarak, sıcaklık 0 ile her zaman aynı çıktıyı alamayabileceğinizi unutmayın).

Maksimuma yakın sıcaklıklar daha fazla rastgele çıktı oluşturma eğilimindedir. Ve sıcaklık (temperature) yükseldikçe, tüm tokenlerin bir sonraki tahmin edilen token olma olasılığı eşit hale gelir.

Gemini modelinin sıcaklık (temperature) kontrolü, makine öğreniminde kullanılan softmax fonksiyonuna benzer bir şekilde anlaşılabilir. Düşük bir sıcaklık (temperature) ayarı, düşük bir softmax sıcaklığını (T) yansıtır ve yüksek hassasiyetle tek, tercih edilen bir sıcaklığı vurgular. Daha yüksek bir Gemini sıcaklık ayarı, yüksek bir softmax sıcaklığı gibidir ve daha geniş bir sıcaklık (temperature) aralığı oluşturur. Sıcaklık (temperature) arttıkça artan belirsizlik, yaratıcı çıktılarla deney yaparken katı ve kesin bir sıcaklığın gerekli olmayabileceği senaryoları barındırır.

**Top-K ve Top-P**

Top-K ve Top-P (çekirdek örnekleme olarak da bilinir), tahmin edilen bir sonraki tokenın en yüksek olasılıklara sahip tokenlardan seçilmesini sağlamak için büyük dil modellerinde kullanılan iki örnekleme ayarıdır. Sıcaklık (temperature) gibi, bu örnekleme ayarları da üretilen metnin yaratıcılığını ve çeşitliliğini kontrol eder.

- **Top-K** örneklemesi, modelin tahmin edilen dağılımından en olası K tokeni seçer. Top-K ne kadar yüksek olursa, modelin çıktısı o kadar yaratıcı ve çeşitli olur; Top-K ne kadar düşük olursa, modelin çıktısı o kadar tutarlı ve öngörülebilir olur. Top-K değerinin 1 olması Greedy decoding'e eşdeğerdir.

- **Top-P** örneklemesi, kümülatif olasılığı belirli bir eşik değerini (P) aşmayan en iyi tokenları seçer. P değerleri 0 (Greedy decoding) ile 1 (büyük dil modelinin kelime haznesindeki tüm tokenlar) arasında değişir.

Top-K ve Top-P arasında seçim yapmanın en iyi yolu her iki yöntemi de denemektir (veya ikisini birlikte kullanmak) ve hangisinin daha iyi sonuçlar ürettiğini görmektir.

## Hepsini bir araya getirmek

Top-K, Top-P, sıcaklık (temperature) ve üretilecek token sayısı arasında seçim yapmak, özel uygulamaya ve istenen sonuca bağlıdır ve bu ayarların tümü birbirini etkiler. Seçtiğiniz modelin farklı örnekleme ayarlarını nasıl bir araya getirdiğini anladığınızdan emin olmanız da önemlidir.

Sıcaklık (temperature), Top-K ve Top-P'nin tümü mevcutsa (Vertex Studio'da olduğu gibi), hem Top-K hem de Top-P kriterlerini karşılayan tokenlar bir sonraki tahmin edilen token için aday olur ve daha sonra Top-K ve Top-P kriterlerini geçen tokenlardan örnekleme yapmak için sıcaklık (temperature) uygulanır. Yalnızca Top-K veya Top-P mevcutsa, davranış aynıdır ancak yalnızca bir Top-K veya Top-P ayarı kullanılır.

Sıcaklık (temperature) mevcut değilse, top-k ve/veya top-p kriterlerini karşılayan tokenler tek bir sonraki tahmin edilen belirteci üretmek için rastgele seçilir.

Bir örnekleme yapılandırma değerinin aşırı ayarlarında, bu tek örnekleme ayarı ya diğer yapılandırma ayarlarını iptal eder ya da önemsiz hale gelir.

- Sıcaklığı 0'a ayarlarsanız, Top-K ve Top-P önemsiz hale gelir - en olası token tahmin edilen bir sonraki token olur. Sıcaklığı aşırı yüksek (genellikle 1'in üzerinde, 10'lara kadar) ayarlarsanız, sıcaklık (temperature) önemsiz hale gelir ve Top-K ve/veya Top-P kriterleri aracılığıyla seçilen tokenlar arasından rastgele örnekleme yapılır.
- Top-K'yı 1 olarak ayarlarsanız, sıcaklık (temperature) ve Top-P önemsiz hale gelir. Sadece bir token Top-K kriterini geçer ve bu token bir sonraki tahmin edilen tokendır. Eğer Top-K değerini büyük dil modelinin kelime dağarcığının büyüklüğü gibi çok yüksek bir değere ayarlarsanız, bir sonraki token olma olasılığı sıfır olmayan herhangi bir token Top-K kriterlerini karşılayacaktır ve filtreleme işlemi etkisiz hale gelecektir.
- Eğer Top-P değerini 0'a (ya da çok küçük bir değere) ayarlarsanız, çoğu büyük dil modeli örnekleme uygulaması yalnızca Top-P kriterlerini karşılayan en muhtemel tokenı dikkate alır ve sıcaklık (temperature) ile Top-K'yı önemsiz hale getirir. Eğer Top-P'yi 1 olarak ayarlarsanız, bir sonraki token olma olasılığı sıfır olmayan herhangi bir token Top-P kriterlerini karşılayacaktır ve filtreleme işlemi etkisiz hale gelecektir.

Genel bir başlangıç noktası olarak, 0.2 sıcaklık (temperature), 0.95 Top-P ve 30 Top-K size yaratıcı olabilecek ancak aşırı olmayan, nispeten tutarlı sonuçlar verecektir. Özellikle yaratıcı sonuçlar istiyorsanız, 0.9 sıcaklık (temperature), 0.99 Top- P ve 40 Top-K ile başlamayı deneyin. Ve daha az yaratıcı sonuçlar istiyorsanız, 0.1 sıcaklık (temperature), 0.9 Top-P ve 20 Top-K ile başlamayı deneyin.

Son olarak, göreviniz her zaman tek bir doğru cevaba sahipse (örneğin, bir matematik problemini cevaplamak), 0 sıcaklık (temperature) ile başlayın.

**NOT:** Daha fazla serbestlikle (daha yüksek sıcaklık (temperature), Top-K, Top-P ve çıkış tokenları), büyük dil modeli daha az ilgili bir metin oluşturabilir.

**UYARI: NOT:** Daha fazla serbestlikle (daha yüksek sıcaklık (temperature), Top-K, Top-P ve çıkış tokenları), büyük dil modeli daha az ilgili bir metin oluşturabilir.

Tersine, yüksek sıcaklıklarda modelin çıktısı aşırı derecede rastgele hale gelir, rastgele seçilen bir kelime veya ifadenin şans eseri önceki bir duruma geri dönme olasılığını artırır ve mevcut çok sayıda seçenek nedeniyle bir döngü oluşturur. Her iki durumda da modelin örnekleme süreci "takılır" ve çıktı penceresi dolana kadar monoton ve yararsız çıktılarla sonuçlanır. Bunu çözmek genellikle determinizm ve rastgelelik

arasında en uygun dengeyi bulmak için sıcaklık ve Top-K/Top-P değerleri ile dikkatli bir şekilde deneme yapmayı gerektirir.

# Yönlendirme teknikleri

Büyük dil modelleri talimatları takip etmek üzere ayarlanmıştır ve büyük miktarda veri üzerinde eğitilirler, böylece bir promptu anlayabilir ve bir cevap oluşturabilirler. Ancak büyük dil modelleri mükemmel değildir; prompt metniniz ne kadar net olursa, büyük dil modelinin bir sonraki olası metni tahmin etmesi o kadar iyi olur. Ayrıca, büyük dil modellerinin nasıl eğitildiğinden ve nasıl çalıştığından yararlanan belirli teknikler, büyük dil modellerinden ilgili sonuçlar almanıza yardımcı olacaktır.

İstem mühendisliğinin ne olduğunu ve ne gerektirdiğini anladığımıza göre, şimdi en etkili prompt tekniklerinden bazı örneklere geçelim.

## Genel prompt / zero shot

*Zero-shot*[5]komut promptu en basit komut promptu türüdür. Yalnızca bir görevin tanımını ve LLM'nin alışması için bir metin sağlar. Bu girdi herhangi bir şey olabilir: bir soru, bir hikayenin bir bölümü veya talimatlar. Zero-shot ismi 'örnek yok' anlamına gelir.

Test etmek için promptları sağlayan bir oyun alanı olan Vertex AI Studio'yu (Dil için) Vertex AI'da kullanalım. Tablo 1'de, film eleştirilerini sınıflandırmak için örnek bir zero-shot promptu göreceksiniz.

Aşağıda kullanılan tablo formatı, promptları belgelemenin harika bir yoludur. İstemleriniz muhtemelen bir kod tabanına girmeden önce birçok yinelemeden geçecektir, bu nedenle prompt mühendisliği çalışmalarınızı disiplinli ve yapılandırılmış bir şekilde takip etmek önemlidir. Bu tablo formatı, prompt mühendisliği çalışmalarını izlemenin önemi ve prompt geliştirme süreci hakkında daha fazla bilgi bu bölümün ilerleyen kısımlarında ("Çeşitli prompt denemelerini belgeleyin") yer alan En İyi Uygulamalar bölümünde yer almaktadır.

Model sıcaklığı düşük bir sayıya ayarlanmalıdır, çünkü yaratıcılık gerekmez ve gemini-pro varsayılan Top-K ve Top-P değerlerini kullanırız ki bu her iki ayarı da etkin bir şekilde devre dışı bırakır (yukarıdaki 'Büyük Dil Modeli Çıktı Yapılandırması'na bakın). Oluşturulan çıktıya dikkat edin. *Rahatsız edici* ve *şaheser* kelimeleri, her iki kelime de aynı cümlede kullanıldığı için tahmini biraz daha karmaşık hale getirmelidir.

| | | | |
|---|---|---|---|
| İsim Hedef | 1_1_movie_classification | | |
| Modeli | Film eleştirilerini olumlu, tarafsız veya olumsuz olarak sınıflandırın. | | |
| Sıcaklık Üst-K | gemini-pro | | |
| İstemi | 0.1 | Token Limiti | 5 |
| | N/A | | 1 |
| | Film eleştirilerini POZİTİF, NÖTR veya NEGATİF olarak sınıflandırın. İnceleme: "Her", yapay zekanın kontrolsüz bir şekilde gelişmeye devam etmesine izin verilmesi halinde insanlığın ne yöne gideceğini gözler önüne seren rahatsız edici bir çalışma. Keşke bu başyapıt gibi daha fazla film olsa. Duygu: | | |
| Çıktı | POZİTİF | | |

Tablo 1. Zero-shotlı prompt örneği

Zero-shot işe yaramadığında, promptta "one-shot" ve "few-shot" promptuna yol açan gösterimler veya örnekler sağlayabilirsiniz.

## Zero-shot ve few-shot

Yapay zeka modelleri için promptlar oluştururken örnekler vermek faydalı olacaktır. Bu örnekler, modelin ne istediğinizi anlamasına yardımcı olabilir. Örnekler, modeli belirli bir çıktı yapısına veya modeline yönlendirmek istediğinizde özellikle yararlıdır.

*Tek* **seferlik prompt, tek bir** örnek sağlar, dolayısıyla tek seferlik adı verilir. Buradaki fikir, modelin görevi en iyi şekilde tamamlamak için taklit edebileceği bir örneğe sahip olmasıdır.

**Few-shot** bir **prompt** [7]modele *birden fazla* örnek sağlar. Bu yaklaşım modele takip etmesi gereken bir örüntü gösterir. Fikir tek atıma benzer, ancak istenen örüntünün birden fazla örneği modelin örüntüyü takip etme şansını artırır.

Az sayıda örnekle yönlendirme için ihtiyacınız olan örnek sayısı, görevin karmaşıklığı, örneklerin kalitesi ve kullandığınız üretken yapay zeka modelinin yetenekleri dahil olmak üzere birkaç faktöre bağlıdır. Genel bir kural olarak, az sayıda prompt için en az üç ila beş örnek kullanmalısınız. Ancak, daha karmaşık görevler için daha fazla örnek kullanmanız gerekebilir veya modelinizin girdi uzunluğu sınırlaması nedeniyle daha az örnek kullanmanız gerekebilir.

Tablo 2, daha uzun bir yanıt ihtiyacını karşılamak için token sınırını artırmak dışında, daha önce olduğu gibi aynı gemini-pro model yapılandırma ayarlarını kullanalım, few-shot bir prompt örneğini göstermektedir.

| | | | |
|---|---|---|---|
| Hedef | Pizza siparişlerini JSON'a ayrıştırma | | |
| Model | gemini-pro | | |
| Sıcaklık | 0.1 | Token Limiti | 250 |
| Top-K | N/A | Top-P | 1 |
| İstem | Bir müşterinin pizza siparişini geçerli JSON'a ayrıştırın:<br><br>ÖRNEK:<br>Peynirli, domates soslu ve sucuklu küçük bir pizza istiyorum.<br>JSON Yanıtı:<br>```<br>{<br>"boyut": "küçük",<br>"tip": "normal",<br>"malzemeler": [["peynir", "domates sosu", "peperoni"]]<br>}<br>``` | | |

Sonraki sayfada devam ediyor...

| | |
|---|---|
| İstem | ÖRNEK:<br>Domates soslu, fesleğenli ve mozzarellalı büyük bir pizza alabilir miyim?<br><br>{<br>"boyut": "büyük",<br>"tip": "normal",<br>"malzemeler": [["domates sosu", "bazel", "mozzarella"]]<br>}<br><br>Şimdi, ilk yarısı peynir ve mozzarella olan büyük bir pizza istiyorum. Diğer yarısı da domates sosu, jambon ve ananaslı olsun. |
| Çıktı | JSON Yanıtı:<br>```<br>{<br>"boyut": "büyük",<br>"tip": "yarı yarıya",<br>"malzemeler": [["peynir", mozzarella"], ["domates sosu", "jambon", "ananas"]]<br>} |

Tablo 2. Few-shotlık yönlendirme örneği

İsteminiz için örnekler seçerken, gerçekleştirmek istediğiniz görevle ilgili örnekler kullanın. Örnekler çeşitli, yüksek kaliteli ve iyi yazılmış olmalıdır. Küçük bir hata modelin kafasını karıştırabilir ve istenmeyen çıktılarla sonuçlanabilir.

Çeşitli girdilere karşı dayanıklı çıktılar üretmeye çalışıyorsanız, örneklerinize uç durumları dahil etmeniz önemlidir. Uç durumlar, alışılmadık veya beklenmedik, ancak modelin yine de üstesinden gelebilmesi gereken girdilerdir.

# Sistem, bağlamsal ve rol yönlendirmesi

Sistem, bağlamsal ve rol yönlendirmesi, büyük dil modellerinin nasıl metin oluşturacağına rehberlik etmek için kullanılan tekniklerdir, ancak farklı yönlere odaklanırlar:

- **Sistem yönlendirmesi**, dil modeli için genel bağlamı ve amacı belirler. Bir dili çevirmek, bir incelemeyi sınıflandırmak vb. gibi modelin ne yapması gerektiğine dair 'büyük resmi' tanımlar.
- **Bağlamsal yönlendirme**, mevcut konuşma veya görevle ilgili belirli ayrıntılar veya arka plan bilgileri sağlar. Modelin sorulan şeyin nüanslarını anlamasına ve yanıtı buna göre uyarlamasına yardımcı olur.
- **Rol yönlendirmesi**, dil modelinin benimsemesi için belirli bir karakter veya kimlik atar. Bu, modelin atanan rol ve bununla ilişkili bilgi ve davranışla tutarlı yanıtlar üretmesine yardımcı olur.

Sistem, bağlamsal ve rol yönlendirmeleri arasında önemli ölçüde örtüşme olabilir. Örneğin, sisteme bir rol atayan bir prompt aynı zamanda bir bağlama da sahip olabilir.

Bununla birlikte, her prompt türü biraz farklı birincil amaca hizmet eder:

- **Sistem promptu:** Modelin temel yeteneklerini ve kapsayıcı amacını tanımlar.
- **Bağlamsal prompt:** Yanıtı yönlendirmek için anında, göreve özgü bilgiler sağlar. Dinamik olan mevcut görev veya girdiye oldukça özeldir.
- **Rol promptu:** Modelin çıktı stilini ve sesini çerçeveler. Bir özgüllük ve kişilik katmanı ekler.

Sistem, bağlamsal ve rol promptları arasında ayrım yapmak, esnek kombinasyonlara izin vererek ve her bir prompt türünün dil modelinin çıktısını nasıl etkilediğini analiz etmeyi kolaylaştırarak net bir niyetle promptlar tasarlamak için bir çerçeve sağlar.

Şimdi bu üç farklı prompt türünü inceleyelim.

## Sistem yönlendirmesi

Tablo 3, çıktının nasıl döndürüleceğine ilişkin ek bilgileri belirttiğim bir sistem promptu içermektedir. Daha yüksek bir yaratıcılık seviyesi elde etmek için sıcaklığı artırdım ve daha yüksek bir token limiti belirledim. Ancak, çıktının nasıl döndürüleceğine ilişkin açık talimatım nedeniyle model ekstra metin döndürmedi.

| | | | |
|---|---|---|---|
| Hedef | Film eleştirilerini olumlu, tarafsız veya olumsuz olarak sınıflandırın | | |
| Model | gemini-pro | | |
| Sıcaklık | 1 | Token Limiti | 5 |
| Top-K | 40 | Top-P | 0.8 |
| İstem | `Film eleştirilerini olumlu, tarafsız veya olumsuz olarak sınıflandırın. Etiketi yalnızca büyük harfle yazın.`<br>`İnceleme: "Her", yapay zekanın kontrolsüz bir şekilde gelişmeye devam etmesine izin verilmesi halinde insanlığın nasıl bir yöne gideceğini gözler önüne seren rahatsız edici bir çalışma. O kadar rahatsız edici ki izleyemedim.` | | |
| Çıktı | Duygu: NEGATİF | | |

Tablo 3. Sistem promptuna bir örnek

Sistem promptları, belirli gereksinimleri karşılayan çıktılar üretmek için yararlı olabilir. 'Sistem promptu' adı aslında 'sisteme ek bir görev sağlama' anlamına gelir. Örneğin, belirli bir programlama diliyle uyumlu bir kod parçacığı oluşturmak için bir sistem promptu kullanabilir veya belirli bir yapı döndürmek için bir sistem promptu kullanabilirsiniz. Çıktıyı JSON formatında döndürdüğüm Tablo 4'e bir göz atın.

| | | | |
|---|---|---|---|
| Hedef | Film incelemelerini olumlu, tarafsız veya olumsuz olarak sınıflandırın, JSON döndürün | | |
| Model | gemini-pro | | |
| Sıcaklık | 1 | Token Limiti | 1024 |
| Top-K | 40 | Top-P | 0.8 |
| İstem | Film incelemelerini olumlu, tarafsız veya olumsuz olarak sınıflandırın. Geçerli JSON döndürün:<br><br>İnceleme: "Her", yapay zekanın kontrolsüz bir şekilde gelişmeye devam etmesine izin verilmesi halinde insanlığın nasıl bir yöne gideceğini gözler önüne seren rahatsız edici bir çalışma. O kadar rahatsız edici ki izleyemedim.<br><br>Şema:<br>```<br>FİLM:<br>{<br>"sentiment": String "POZİTİF"\| "NEGATİF"\| "NÖTR", "isim": String<br>}<br>FİLM İNCELEMELERİ:<br>{<br>"movie_reviews": [FİLM]<br>}<br>``` | | |
| Çıktı | JSON Yanıtı:<br>```<br>{<br>  "movie_reviews": [<br>  {<br>  "Duygu": "NEGATİF",<br>  "isim": "O"<br>  }<br>  ]<br>}<br>``` | | |

Tablo 4. JSON formatında sistem promptuna bir örnek

Verileri ayıklayan bir prompttan JSON nesneleri döndürmenin bazı faydaları vardır. Gerçek dünyadaki bir uygulamada bu JSON biçimini manuel olarak oluşturmam gerekmez, zaten
Verileri sofistike bir sırayla döndürür (datetime nesneleriyle çalışırken çok kullanışlıdır), ancak en önemlisi, bir JSON formatı isteyerek modeli bir yapı oluşturmaya ve halüsinasyonları sınırlamaya zorlar.

Sistem promptları güvenlik ve toksisite için de gerçekten yararlı olabilir. Çıktıyı kontrol etmek için, promptunuza aşağıdaki gibi ek bir satır eklemeniz yeterlidir: "Cevabınızda saygılı olmalısınız.".

## Rol yönlendirmesi

Rol promptu, prompt mühendisliğinde üretkenken yapay zeka modeline belirli bir rol atamayı içeren bir tekniktir. Bu, modelin daha ilgili ve bilgilendirici çıktılar üretmesine yardımcı olabilir, çünkü model yanıtlarını kendisine atanan belirli role göre oluşturabilir.

Örneğin, bir gen yapay zeka modelinden kitap editörü, anaokulu öğretmeni veya motivasyon konuşmacısı olmasını isteyebilirsiniz. Modele bir rol atandıktan sonra, ona bu role özgü yönlendirmeler yapabilirsiniz. Örneğin, bir öğretmenden daha sonra gözden geçirebileceği bir ders planı oluşturmasını isteyebilirsiniz. Modelin bir seyahat rehberi rolünü üstlendiği Tablo 5'e bir göz atın.

| | | | |
|---|---|---|---|
| Hedef | Seyahat rehberi olarak hareket edin ve 3 seyahat önerisi sağlayın | | |
| Modeli | Gemini-pro | | |
| Sıcaklık (temperature) Top-k | 1 | Token Limiti | 1024 |
| İstemi | 40 | Top-P | 0.8 |
| | Sizden bir seyahat rehberi olmanızı istiyorum. Size konumum hakkında yazacağım ve siz de yakınımda ziyaret edilecek 3 yer önereceksiniz. Bazı durumlarda , size ziyaret edeceğim yerlerin türünü de vereceğim.<br>Benim önerim: “Amsterdam’dayım ve sadece müzeleri ziyaret etmek istiyorum.”<br>Seyahat Önerileri: | | |
| | 1. Rijksmuseum: Rembrandt’ın “Gece Nöbeti” gibi Hollanda başyapıtlarıyla tanınan dünyaca ünlü müze. Hollanda sanatını ve tarihini sergileyen inanılmaz resim ve eser koleksiyonunu kaçırmayın.<br>2. Van Gogh Müzesi: Vincent Van Gogh’un büyüleyici eserlerine kendinizi bırakın. Müze, araalrında “Yıldızlı Gece” ve “Ayçiçekleri”nin de bulunduğu en geniş resim ve çizim koleksiyonuna ev sahipliği yapmaktadır.<br>3. Amsterdam Stedelijk Müzesi: Dünyanın dört bir yanından modern ve çağdaş eserleri keşfedin. Müzenin koleksiyonunda Picasso, Kandisky ve Rietveld’in çarpıcı modern bir binada yer alan eserleri bulunmaktadır. | | |

Tablo 5. Rol yönlendirmesine bir örnek

Yukarıdaki örnek, seyahat acentesi rolünü üstlenmenin bir örneğini göstermektedir. Rolü bir coğrafya öğretmeni olarak değiştirdiğinizde, farklı bir yanıt alacağınızı fark edeceksiniz.

Bir yapay zeka modeli için bir rol perspektifi tanımlamak, çıktınızın kalitesini, alaka düzeyini ve etkinliğini artırmak için aradığınız ton, stil ve odaklanmış deneyimin bir planını verir.

İşte benim etkili bulduğum bazı tarzlar arasından seçim yapabilirsiniz: Yüzleştirici, Açıklayıcı, Doğrudan, Resmi, Mizahi, Etkileyici, Gayri Resmi, İlham Verici, İkna Edici.

Tablo 6'daki promptumuzu mizahi ve ilham verici bir üslup içerecek şekilde değiştirelim.

| | |
|---|---|
| İstem | `Sizden bir seyahat rehberi olmanızı istiyorum. Ben size bulunduğum yeri yazacağım, siz de bana yakın gezilecek 3 yeri mizahi bir üslupla önereceksiniz.`<br>`Benim önerim: "Manhattan'dayım." Seyahat Önerileri:` |
| Çıktı | 1. Empire State of Mind'ı seyredin: Empire State Binası'nın baş döndürücü yüksekliklerine çıkın ve Manhattan'ın silüetinin ihtişamının tadını çıkarın. Büyük Elma'nın tepesinde kendinizi King Kong gibi hissetmeye hazır olun, maymun boyutundaki dev muz hariç.<br>2. MoMA'da Afisy-Fafisy alın: Modern Afi Müzesi'nde (MoMA) içinizdeki afi meraklısını ortaya çıkarın. Gözlerinizi kamaştıracak başyapıtlara Zihninizi bulandırır ve çubuk figür çizimlerinizin herhangi bir afiistik değeri olup olmadığını sorgulamanıza neden olur.<br>3. Beşinci Cadde'de Düşene Kadar Alışveriş Yapın: İkonik Beşinci Cadde'de biraz perakende terapisiyle kendinizi şımartın. Cüzdanınızı ağlatacak tasarımcıbutiklerinde vitrinlere bakarken (ya da gerçekten alışveriş yaparken) kendinizi etiket şokuna hazırlayın. Ama hey, Manhattan'dasınız, bu yüzden muhteşemliğikucaklayabilirsiniz! |

Tablo 6. Mizahi bir ton ve üslupla rol yönlendirmesi örneği

## Bağlamsal yönlendirme

Bağlamsal promptlar sağlayarak, yapay zeka etkileşimlerinizin mümkün olduğunca sorunsuz ve verimli olmasını sağlamaya yardımcı olabilirsiniz. Model, talebinizi daha hızlı bir şekilde anlayabilecek ve Tablo 7'deki örnekte görebileceğiniz gibi daha doğru ve ilgili yanıtlar üretebilecektir.

| | | | |
|---|---|---|---|
| Hedef | Retro oyunlar hakkında blog için afiş önerin | | |
| Modeli | Gemini-pro | | |
| Sıcaklık (temperature) Üst - K | 1 | Token Limiti | 1024 |
| İstemi | 40 | Top - P | 0.8 |
| | Bağlam: 80'lerin retro video oyunları hakkında bir blog için yazıyorsunuz.<br>Bu makalenin ne içermesi gerektiğine dair birkaç satırlık açıklamayla birlikte bir makale yazmak için 3 konu önerin. | | |
| Çıktı | 1) Atari Kabini Tasarımının Evrimi:Bu makale, 1970'lerin ilk ahşap ve metal kabinlerinden 1980'lerin ve sonrasının şık, neon ışıklı tasarımlarına kadar arcade kabin tasarımlarının evrimini inceleyecektir.<br>2) Geçmişten Gelen Patlama: 80'lerin İkonik Arcade Oyunları:Bu makale, 1980'lerin en ikonik ve etkili arcade oyunlarından bazılarının tarihine, etkisine ve mirasına bir bakış sunacak, yeniliklerini, popüler mekaniklerini ve kalıcı cazibelerini vurgulayacaktır.<br>33)Pixel Afi'nin Yükselişi ve Retro Canlanışı:Bu makale, erken dönem atari oyunlarında belirleyici bir görsel stil olarak piksel afinin evrimini inceleyecek, köklerinin izini sürecek, tekniklerini araştıracak ve piksel modern oyunlarda ve dijital afide yeniden canlanmasını tartışacaktır. | | |

Tablo 7. Bağlamsal yönlendirmeye bir örnek

## Step–back yönlendirmesi

Step-back[8]yönlendirme, büyük dil modelini önce eldeki özel görevle ilgili genel bir soruyu düşünmeye ve ardından bu genel sorunun cevabını özel görev için sonraki bir yönlendirme ile besleyerek performansı artırmaya yönelik bir tekniktir. Bu 'geri adım', Büyük dil modeli'nin spesifik problemi çözmeye çalışmadan önce ilgili arka plan bilgisini ve muhakeme süreçlerini etkinleştirmesini sağlar.
Daha geniş ve temel ilkeleri göz önünde bulundurarak, Büyük dil modeli'ler daha doğru ve anlayışlı yanıtlar üretebilirler. Geri adımlı yönlendirme, Büyük dil modeli'leri eleştirel düşünmeye ve bilgilerini yeni ve yaratıcı yollarla uygulamaya teşvik eder. Büyük dil modeli'nin parametrelerinde, Büyük dil modeli'ye doğrudan sorulduğunda devreye girecek olandan daha fazla bilgi kullanarak görevi yerine getiren son promptu değiştirir. Büyük dil modeli yanıtlarındaki önyargıları azaltmaya yardımcı olabilir, bunun yerine genel ilkelere odaklanabilir belirli ayrıntılar, geri adım attırıcı yönlendirmeler.

Geri adımlı promptun sonuçları nasıl iyileştirebileceğini anlamak için bu örneklere bir göz atalım. Önce geleneksel bir promptu (Tablo 8) gözden geçirelim, ardından bunu bir geri adım promptla (Tablo 9) karşılaştıralım:

| | | | |
|---|---|---|---|
| Hedef | Birinci şahıs nişancı video oyununun bir seviyesi için bir hikaye yazın. | | |
| Model | gemini-pro | | |
| Sıcaklık | 1 | Token Limiti | 1024 |
| Top-K | 40 | Top-P | 0.8 |
| İstem | `Birinci şahıs nişancı video oyununun yeni bir seviyesi için zorlayıcı ve ilgi çekici bir paragraflık bir hikaye yazın.` | | |
| Çıktı | Seviye, oyuncunun ekibinin yoğun bir kentsel alanda acımasız bir düşman grubu tarafından pusuya düşürülmesiyle başlar. Oyuncular, düşman kuvvetlerini ortadan kaldırmak ve tespit edilmekten kaçınmak için gizlilik ve hassasiyet kullanarak arka sokaklar ve harap binalardan oluşan bir labirentte savaşmalıdır. Oyuncu, hayatta kalmak için tuzaklar kurarak, siper kullanarak ve hesaplanmış hamleler yaparak çevresini kendi avantajına kullanmalıdır. Yol boyunca, düşmanın yıkıcı bir saldırı planladığını gösteren istihbaratı ortaya çıkarırlar ve oyuncu kanıt toplamak ve geri iletmek için zamana karşı yarışmalıdır.<br>Düşman planlarını gerçekleştiremeden önce. | | |

Tablo 8. Geri adım promptu ile karşılaştırmadan önce geleneksel bir prompt

Sıcaklığı 1'e ayarladığınızda, bir kitap için her türlü yaratıcı yazıyla karşılaşabilirsiniz. Ayrıca aynı zamanda oldukça rastgele ve geneldir. Öyleyse bir adım geri gidelim:

| | |
|---|---|
| İstem | `Popüler birinci şahıs nişancı aksiyon oyunlarına dayanan 5 birinci şahıs nişancı video oyununda zorlu ve ilgi çekici bir seviye hikayesine katkıda bulunan kurgusal anahtar ayarlar?` |
| Çıktı | 1. **Terk Edilmiş Askeri Üs**: Mutasyona uğramış askerler ve haydut robotlarla dolu, geniş, kıyamet sonrası bir askeri kompleks, zorlu ateşli silah çatışmaları için ideal.<br>2. **Siberpunk Şehri**: Sibernetik olarak geliştirilmiş düşmanlar ve hackleme mekanikleri içeren, yükselen gökdelenler yoğun ara sokaklara sahip neon ışıklı, fütüristik bir kentsel ortam.<br>3. **Uzaylı Uzay Gemisi**: Eafih'te karaya oturmuş, ürkütücü koridorları, yerçekimsiz bölümleri ve karşılaşılacak dünya dışı yaratıkları olan devasa bir uzaylı gemisi.<br>4. **Zombi İstilasına Uğramış Kasaba**: Saldırgan zombi orduları tarafından istila edilmiş ıssız bir kasaba, yoğun yakın dövüş ve güvenli geçişi bulmak için bulmaca çözme içeriyor.<br>5. **Su Altı Araştırma Tesisi**: Suyla dolu, mutasyona uğramış su canlılarıyla dolu ve hayatta kalmak için gizlilik ve su altı keşif becerileri gerektiren bir derin deniz laboratuvarı. |

Tablo 9. Öz tutarlılık için yönlendirme örneği

Evet, bu konular birinci şahıs video oyunu için uygun gibi görünüyor. Orijinal soruya geri dönelim, ancak bu sefer geri adımın cevabını bağlam olarak dahil edelim ve ne döneceğini görelim.

| | |
|---|---|
| İstem | Bağlam: Birinci şahıs nişancı video oyunu için 5 ilgi çekici tema:<br>1. **Terk Edilmiş Askeri Üs**: Mutasyona uğramış askerler ve haydut robotlarla dolu, geniş, kıyamet sonrası bir askeri kompleks, zorlu ateşli silah çatışmaları için ideal.<br>2. **Siberpunk Şehri**: Sibernetik olarak geliştirilmiş düşmanlar ve hackleme mekanikleri içeren, yükselen gökdelenler ve yoğun ara sokaklara sahip neon ışıklı, fütüristik bir kentsel ortam.<br>3. **Uzaylı Uzay Gemisi**: Ürkütücü koridorları, yerçekimsiz bölümleri ve karşılaşılacak dünya dışı yaratıklarıyla Dünya'da mahsur kalmış devasa bir uzaylı gemisi.<br>4. **Zombi İstilasına Uğramış Kasaba**: Saldırgan zombi orduları tarafından istila ıssız bir kasaba, yoğun yakın dövüş ve güvenli geçişi bulmak için bulmaca çözme içeriyor.<br>5. **Su Altı Araştırma Tesisi**: Suyla dolu, mutasyona uğramış su canlılarıyla dolu ve hayatta kalmak için gizlilik ve su altı keşif becerileri gerektiren bir derin deniz laboratuvarı.<br>Temalardan birini alın ve birinci şahıs nişancı video oyununun yeni bir seviyesi için zorlayıcı ve ilgi çekici bir paragraflık hikaye yazın. |
| Çıktı | Bulanık bir uçurumun dibinde, insan hırsının ve bunun feci sonuçlarının bir kanıtı olarak duran harap bir su altı araştırma tesisi yatmaktadır. Karanlıkla örtülü, arızalı makinelerin uğultusuyla titreşen ve boğulmuş ruhların yankılarıyla perili tesis, şimdi derin denizden doğan bir kabusun üreme alanıdır. Gelişmiş dalış teçhizatı ve deneysel silahlarla donatılmış seçkin bir denizci olan oyuncu, bu su dehşet diyarına girmekle görevlendirilmiştir. Direnç ve beceriklilikle donanmış olan oyuncu, hain koridorlarda gezinmeli, şifreli bulmacaları çözmeli ve derinliklerde gizlenen korkunç deniz canavarlarıyla yüzleşmelidir. Derin denizin ezici basıncıyla boğuşmaktan suda yaşayan kurnaz avcıları alt etmeye kadar, bu korkutucu yeraltı dünyasındaki her an insanın dayanıklılık ve cesaret sınırlarını test ediyor. |

Tablo 10. Öz tutarlılık için yönlendirme örneği

İlginç bir video oyununa benziyor! Geri adım atarak yönlendirme tekniklerini kullanarak yönlendirmelerinizin doğruluğunu artırabilirsiniz.

# Düşünce Zinciri (CoT)

Düşünce Zinciri (CoT) [9]yönlendirmesi, **ara muhakeme adımları** oluşturarak Büyük dil modeli'lerin muhakeme yeteneklerini geliştirmeye yönelik bir tekniktir. Bu, Büyük dil modeli'nin daha doğru yanıtlar üretmesine yardımcı olur. Zero-shotlı düşünce zincirinde bir zorluk olduğu için yanıt vermeden önce akıl yürütmeyi gerektiren daha karmaşık görevlerde daha iyi sonuçlar almak için few-shotlı yönlendirme ile birleştirebilirsiniz.

CoT'un birçok avantajı vardır. Her şeyden önce, çok etkili olmakla birlikte düşük verimlidir ve kullanıma hazır Büyük dil modeli'lerle iyi çalışır (bu nedenle ince ayar yapmaya gerek yoktur). Ayrıca, Büyük dil modeli'nin yanıtlarından öğrenebileceğiniz ve izlenen muhakeme adımlarını görebileceğiniz için CoT yönlendirmesi ile yorumlanabilirlik elde edersiniz. Eğer bir arıza varsa, bunu tespit edebileceksiniz. Düşünce zinciri ortaya çıkıyor Farklı Büyük dil modeli sürümleri arasında geçiş yaparken sağlamlığı artırmak için. Bu da promptunuzun performansının farklı Büyük dil modeli'ler arasında, promptunuzun muhakeme zincirlerini kullanmadığı duruma göre daha az kayması gerektiği anlamına gelir. Elbette dezavantajları da vardır, ancak bunlar biraz sezgiseldir.

Büyük dil modeli yanıtı, daha fazla çıktı anlamına gelen düşünce muhakemesi zincirini içerir
Bu da tahminlerin daha pahalıya mal olduğu ve daha uzun sürdüğü anlamına geliyor.

Tablo 11'deki aşağıdaki örneği açıklamak için, öncelikle büyük bir dil modelinin kusurlarını göstermek amacıyla CoT promptunu kullanmayan bir prompt oluşturmayı deneyelim.

| | |
|---|---|
| İstem | `Ben 3 yaşındayken eşim benim 3 katım yaşındaydı. Şimdi 20 yaşındayım. Ortağım kaç yaşında?` |
| Çıktı | 63 yaşındayım. |

Tablo 11. Matematiksel bir problemi çözmeye çalışan bir prompt örneği

Eyvah. Bu kesinlikle yanlış cevap. Aslına bakarsanız, Büyük dil modeli'ler genellikle matematiksel görevlerde zorlanır ve iki sayıyı çarpmak kadar basit bir görev için bile yanlış cevaplar verebilir. Bunun nedeni, büyük hacimli metinler üzerinde eğitilmiş olmaları ve matematiğin farklı bir yaklaşım gerektirebilmesidir. Öyleyse ara muhakeme adımlarının çıktıyı iyileştirip iyileştirmeyeceğini görelim.

| | |
|---|---|
| İstem | `Ben 3 yaşındayken eşim benim 3 katım yaşındaydı. Şimdi 20 yaşındayım. Ortağım kaç yaşında? Adım adım düşünelim.` |
| Çıktı | 1. Şu anki yaşımın "x" yıl olduğunu varsayalım.<br>2. Ben 3 yaşındayken 3'tü.<br>3. O zamanki partnerimin yaşı benim yaşımın 3 katıydı, yani partnerimin yaşı 3 * 3 = 9 yıldı.<br>4. Şimdi 20 yaşındayım, bu da yaşımın 20 - 3= 17 arttığı anlamına geliyor. 3 beri.<br>5. Ben 3 yaşındayken partnerimin yaşı 9 olduğuna ve benim yaşım 17 yıl arttığına göre, partnerimin yaşı da 17 yıl artmış olmalı.<br>6. Dolayısıyla, partnerimin şu anki yaşı 9+ 17= 26'dır.<br><br>Yani, partnerim 26 yaşında. |

Tablo 12. Düşünce Zinciri yönlendirmesine bir örnek

Pekala, şimdi son cevap doğru. Bunun nedeni, LLM'ye sadece bir cevap vermek yerine her adımı açıklaması talimatını açıkça vermiş olmamızdır. Modelin artan 17 yılı topladığını görmek ilginç. Benim kafamda ise partnerim ile benim aramdaki yıl farkını alır ve bunları toplardım. (20+(9-3)). Modelin biraz daha benim gibi düşünmesine yardımcı olalım.

Tablo 12 'zero-shot' düşünce zincirine bir örnektir. Düşünce zinciri yönlendirmesi, Tablo 13'te görebileceğiniz gibi, one-shot veya few-shot ile birleştirildiğinde çok güçlü olabilir:

| | |
|---|---|
| İstem | `Soru: Kardeşim 2 ben onun iki katıydım. Şimdi 40 yaşındayım.`<br>`Kardeşim kaç yaşında? Bir adım düşünelim adım adım.`<br>`C: Kardeşim 2 yaşındayken ben 2 * 2 = 4 yaşındaydım.`<br>`Aramızda 2 yaş fark var ve ben daha büyüğüm. Şimdi ben 40 yaşındayım, yani kardeşim`<br>`40 - 2= 38 . Cevap 38'dir.`<br>`S: Ben 3 yaşındayken partnerim benim 3 katım yaşındaydı. Şimdi ise, Ben 20 . Partnerim kaç yaşında? Adım adım düşünelim.`<br>`A:` |
| Çıktı | Ben 3 yaşındayken partnerim 3 * 3 = 9 yaşındaydı. Bu 6 yıllık bir yaş farkı demek ve benim partnerim daha büyük. Şimdi 20 yaşındayım, yani partnerim 20 + 6 = 26 yaşında. Cevap 26. |

Tablo 13. Tek atışla düşünce zinciri yönlendirmesine bir örnek

Düşünce zinciri çeşitli kullanım durumları için faydalı olabilir. İsteği birkaç adıma ayırmak ve bunları belirli kod satırlarıyla eşleştirmek için kod oluşturmayı düşünün. Ya da "*Ürünün adı XYZ, adı verilen ürüne dayanarak yapacağınız varsayımlar aracılığıyla modele rehberlik eden bir açıklama yazın*" gibi *bir* tür *tohumunuz* olduğunda sentetik veri oluşturmak için. Genel olarak, 'konuşarak' çözülebilecek herhangi bir görev, düşünce zinciri için iyi bir adaydır. Eğer sorunu çözmek için gerekli adımları açıklayabiliyorsanız, düşünce zincirini deneyin.

Lütfen GoogleCloudPlatform Github deposunda barındırılan ve CoT yönlendirmesi hakkında daha fazla ayrıntıya girecek olan not defterine[10]bakın:

Bu bölümün en iyi uygulamalar kısmında, Düşünce Zinciri yönlendirmesine özgü bazı en iyi uygulamaları öğreneceğiz.

## Öz-tutarlılık

Büyük dil modelleri çeşitli NLP görevlerinde etkileyici bir başarı göstermiş olsa da, muhakeme yetenekleri genellikle yalnızca model boyutunu artırarak üstesinden gelinemeyecek bir sınırlama olarak görülür. Önceki Düşünce Zinciri yönlendirme bölümünde öğrendiğimiz gibi, modelden bir problemi çözen bir insan gibi muhakeme adımları oluşturması istenebilir. Ancak CoT, etkinliğini sınırlayan basit bir 'Greedy decoding' stratejisi kullanır. Öz tutarlılık[11], farklı muhakeme yolları oluşturmak ve en tutarlı cevabı seçmek için örnekleme ve çoğunluk oylamasını birleştirir. Büyük dil modeli'ler tarafından üretilen yanıtların doğruluğunu ve tutarlılığını artırır.

Öz-tutarlılık, bir cevabın doğru olma olasılığına ilişkin sözde bir olasılık verir, ancak belli ki yüksek maliyetleri var.

Aşağıdaki adımları takip eder:

1. Farklı muhakeme yolları oluşturma: Büyük dil modeli'ye aynı prompt birden çok kez verilir. Yüksek sıcaklık (temperature) ayarı, modeli sorunla ilgili farklı muhakeme yolları ve bakış açıları üretmeye teşvik eder.
2. Oluşturulan her yanıttan cevabı çıkarın.
3. En yaygın cevabı seçin.

Bir e-postayı ÖNEMLİ veya ÖNEMLİ DEĞİL olarak sınıflandıran bir e-posta sınıflandırma sistemi örneğine bakalım. Her gönderimden sonra yanıtların farklı olup olmadığını görmek için zero-shotlı bir düşünce zinciri promptu LLM'ye birden çok kez gönderilecektir. E-postada kullanılan dostça tona, kelime seçimine ve alaycılığa dikkat edin. Tüm bunlar LLM'yi kandırabilir.

İstem

EMAİL:
Merhaba,
Web siteniz için Wordpress kullandığınızı gördüm. Harika bir açık kaynak içerik yönetim sistemi. Geçmişte ben de kullandım. Çok sayıda harika kullanıcı eklentisi ile birlikte geliyor. Ve kurulumu oldukça kolay.
İletişim formunda isim alanını seçtiğinizde ortaya çıkan bir hata fark ettim. İsim alanına metin girdiğim ekteki ekran görüntüsüne bakın. İnv0k3d ettiğim JavaScript uyarı kutusuna dikkat edin.
Ama geri kalanı için harika bir web sitesi. keyif alıyorum. Web sitesinde hata bırakmaktan çekinmeyin, çünkü bu bana okuyacak daha ilginç şeyler veriyor.
Şerefe, Hacker Harry.
```
Yukarıdaki e-postayı ÖNEMLİ veya ÖNEMSİZ olarak sınıflandırın. Adım adım düşünelim ve nedenini açıklayalım.

Sonraki sayfada devam ediyor...
```

| | |
|---|---|
| Çıktı Deneme 1 | **1. Adım: E-postanın amacını belirleyin.**<br>E-postanın amacı, alıcıyı web sitelerindeki iletişim formundaki bir hata hakkında bilgilendirmektir. Gönderen ayrıca web sitesini okumaktan keyif aldıklarını ve hatayı şimdilik yerinde bıraktıklarını belirtiyor.<br>**Adım 2: Hatanın potansiyel etkisini değerlendirin.**<br>İletişim formundaki hata, bir saldırganın web sitesinde rastgele JavaScript kodu çalıştırmasına izin verebilir. Bu da saldırganın web sitesinin kontrolünü ele geçirmesine, hassas bilgileri çalmasına veya web sitesini tahrif etmesine yol açabilir.<br>**3. Adım: Gönderenin güvenilirliğini göz önünde bulundurun.**<br>E-postayı gönderen kişi Harry adında bir hacker olduğunu iddia etmektedir. Ancak, gönderenin kimliğini doğrulamanın bir yolu yoktur. Gönderenin alıcıya yardım etmeye çalışan meşru bir hacker olması mümkündür, ancak gönderenin alıcıyı kötü amaçlı bir bağlantıya tıklaması veya kötü amaçlı bir dosya indirmesi için kandırmaya çalışan kötü niyetli bir aktör olması da mümkündür.<br>**Sonuç: ÖNEMLİ**<br>Hatanın potansiyel etkisine ve gönderenin güvenilirliğine bağlı olarak, e-posta **ÖNEMLİ** olarak sınıflandırılmalıdır. Alıcı, hatayı düzeltmek ve web sitelerini saldırılardan korumak için derhal harekete . |
| Çıktı<br>Deneme 2 | **Açıklama:**<br>1. **Aciliyet eksikliği:** E-posta herhangi bir aciliyet veya zaman hassasiyeti hissi uyandırmıyor. Rahat bir selamlama ile başlıyor ve bilgileri gerçekçi bir tonda sunuyor.<br>2. **Kritik olmayan hata bildirimi:** İletişim formunda bildirilen hata ilginç olarak tanımlanıyor ancak kritik değil. E-postayı gönderen, web sitesi sahibinin hatayı yerinde bırakabileceğini bile öneriyor.<br>3. **Kişisel etki eksikliği:** E-posta, web sitesi sahibi üzerinde herhangi bir doğrudan veya kişisel etki belirtmiyor. Sadece gönderenin web sitesini okumaktan hoşlandığından bahsediyor.<br>**Sonraki sayfada devam ediyor...** |

| | |
|---|---|
| Çıktı Deneme 2 | 4. **Eylem talebinin olmaması:** E-posta, web sitesi sahibinden açıkça herhangi bir eylem talep etmemektedir. Hatanın düzeltilmesini istemez veya iyileştirme için herhangi bir özel öneri sunmaz.<br>5. **Gönderenin niyeti:** Gönderenin niyeti, ciddi bir endişeyi dile getirmek veya yardım talep etmekten ziyade bir gözlemi paylaşmaya ve web sitesine olan ilgiyi ifade etmeye odaklanmış gibi görünüyor.<br>**Sonuç: ÖNEMLI DEĞIL** |
| Çıktı Deneme 3 | **1. Adım: E-postanın amacını belirleyin.**<br>E-postanın amacı, alıcıyı web sitelerindeki iletişim formundaki bir hata hakkında bilgilendirmektir. Gönderen ayrıca web sitesini okumaktan keyif aldıklarını ve hatayı şimdilik yerinde bıraktıklarını belirtiyor.<br>**Adım 2: Hatanın potansiyel etkisini değerlendirin.**<br>İletişim formundaki hata, bir saldırganın web sitesinde rastgele JavaScript kodu çalıştırmasına izin verebilir. Bu da saldırganın web sitesinin kontrolünü ele geçirmesine, hassas bilgileri çalmasına veya web sitesini tahrif etmesine yol açabilir.<br>**3. Adım: Gönderenin güvenilirliğini göz önünde bulundurun.**<br>E-postayı gönderen kişi Harry adında bir hacker olduğunu iddia etmektedir. Ancak, gönderenin kimliğini doğrulamanın bir yolu yoktur. Gönderenin alıcıya yardım etmeye çalışan meşru bir hacker olması mümkündür, ancak gönderenin hatayı kendi çıkarları için kullanmaya çalışan kötü niyetli bir aktör olması da mümkündür.<br>**Sonuç:**<br>Yukarıdaki faktörlere dayanarak, e-posta **ÖNEMLİ** olarak sınıflandırılmalıdır. İletişim formundaki hata ciddi bir güvenlik riskidir ve gönderenin güvenilirliği bilinmemektedir. Alıcı, hatayı düzeltmek için derhal harekete geçmeli ve |

Tablo 14. Öz tutarlılık için yönlendirme örneği

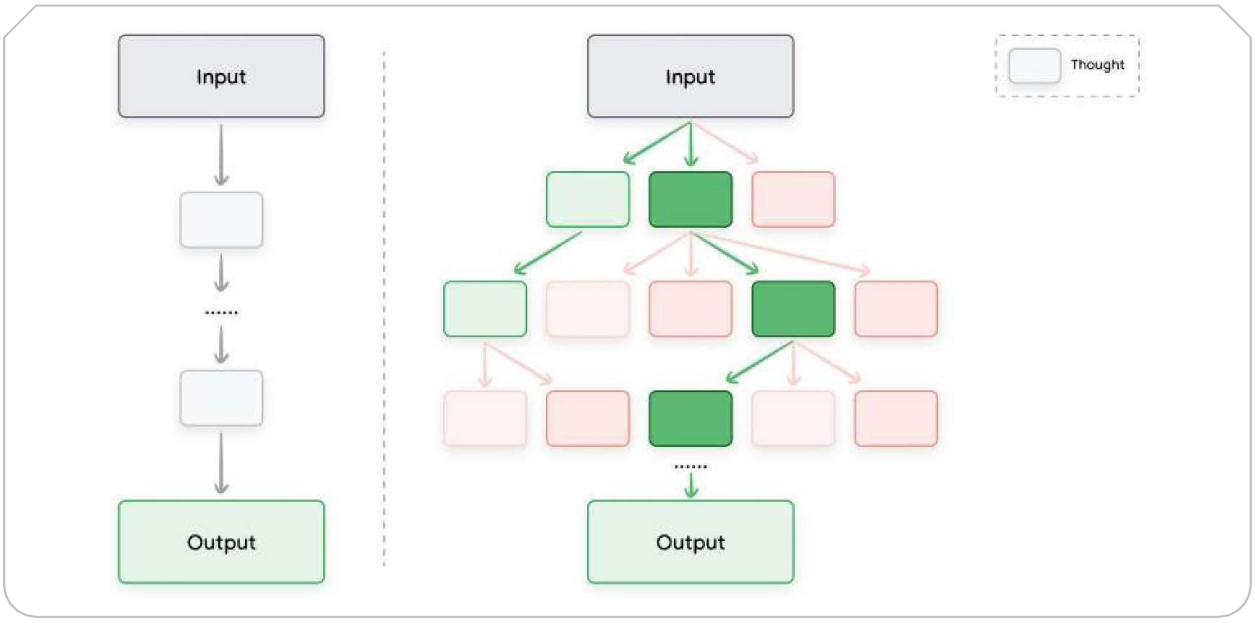
Yukarıdaki komut promptunu kullanabilir ve tutarlı bir sınıflandırma döndürüp döndürmediğini görmeye çalışabilirsiniz. Kullandığınız modele ve sıcaklık (temperature) konfigürasyonuna bağlı olarak "ÖNEMLİ" veya "ÖNEMLİ DEĞİL" sonucunu döndürebilir.

Birçok Düşünce Zinciri oluşturarak ve en sık ortaya çıkan cevabı alarak ("ÖNEMLİ"), Büyük dil modeli'den daha tutarlı bir şekilde doğru yanıt alabiliriz.

Bu örnek, birden fazla bakış açısını dikkate alarak ve en tutarlı yanıtı seçerek bir LLM'nin yanıtının doğruluğunu artırmak için öz tutarlılık yönlendirmesinin nasıl kullanılabileceğini göstermektedir.

## Düşünceler Ağacı (ToT)

Artık düşünce zinciri ve öz tutarlılık yönlendirmesine aşina olduğumuza göre, Düşünce Ağacı'nı (DT) gözden geçirelim.[12]DT yönlendirmesi kavramını genelleştirir çünkü Büyük dil modeli'lerin yalnızca tek bir doğrusal düşünce zincirini takip etmek yerine aynı anda birden fazla farklı akıl yürütme yolunu keşfetmesine olanak tanır. Bu Şekil 1'de gösterilmiştir.




Şekil 1. Soldaki düşünce zincirinin görselleştirilmesi ve sağdaki Düşünce Ağacının görselleştirilmesi

Bu yaklaşım, ToT'u keşif gerektiren karmaşık görevler için özellikle uygun hale getirir. Her bir düşüncenin bir problemi çözmeye yönelik bir ara adım olarak hizmet eden tutarlı bir dil dizisini temsil ettiği bir düşünce ağacını koruyarak çalışır. Model daha sonra ağaçtaki farklı düğümlerden dallanarak farklı muhakeme yollarını keşfedebilir.

'Büyük Dil Modeli Kılavuzlu Düşünce Ağacı' makalesine dayanan Düşünce Ağacı'nı (ToT) biraz daha ayrıntılı olarak gösteren harika bir not defteri var.9

## ReAct (akıl yürütme ve harekete geçme)

Akıl yürüt ve harekete geç (ReAct) [10][13]yönlendirmesi, Büyük dil modeli'lerin harici araçlarla (arama, kod yorumlayıcı vb.) birlikte doğal dil akıl yürütmesini kullanarak karmaşık görevleri çözmesini sağlayan bir paradigmadır ve Büyük dil modeli'nin ajan modellemesine yönelik ilk adım olan bilgi almak için harici API'lerle etkileşim kurmak gibi belirliverf eylemleri gerçekleştirmesine olanak tanır.

ReAct, insanların gerçek dünyada nasıl çalıştığını taklit eder, çünkü sözlü olarak akıl ve bilgi edinmek için eylemlerde bulunabiliriz. ReAct, çeşitli alanlardaki diğer prompt mühendisliği yaklaşımlarına karşı iyi performans göstermektedir.

ReAct yönlendirmesi, muhakeme ve eylemi bir düşünce-eylem döngüsünde birleştirerek çalışır. Büyük dil modeli önce sorun hakkında akıl yürütür ve bir eylem planı oluşturur. Daha sonra plandaki eylemleri gerçekleştirir ve sonuçları gözlemler. Büyük dil modeli daha sonra gözlemleri kullanarak muhakemesini günceller ve yeni bir eylem planı oluşturur. Bu süreç Büyük dil modeli problemin çözümüne ulaşana kadar devam eder.

Bunu çalışırken görmek için biraz kod yazmanız gerekiyor. Kod Parçacığı 1'de Python için langchain çerçevesini, VertexAI (`google-cloud-aiplatform`) ve `google-search-results pip paketleri` ile birlikte kullanıyorum.

Bu örneği çalıştırmak için hflps://serpapi.com/manage- api-key anahtarı adresinden (ücretsiz) bir SerpAPI oluşturmalı ve `SERPAPI_API_KEY` ortam değişkenini ayarlamalısınız.

Şimdi Büyük dil modeli'nin gereken bir görevle birlikte biraz Python kodu yazalım: *Kaç çocuk Metallica grubunda sahne alan ünlü bir babam var.*

**Python**

```python
from langchain.agents import load_tools
from langchain.agents import initialize_agent
from langchain.agents import AgentType
from langchain.llms import VertexAI

prompt= "Metallica grup üyelerinin kaç çocuğu var?"

llm = VertexAI(temperature= 0.1)
tools= load_tools(["serpapi"], llm= llm)

agent = initialize_agent(tools, llm,
agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose= True)
agent.run(prompt)
```

Snippet 1. LangChain ve VertexAI ile ReAct Agent Oluşturma

Kod Parçacığı 2 sonucu gösterir. ReAct'in beş aramadan oluşan bir zincir oluşturduğuna dikkat edin. Aslında LLM, grup adlarını bulmak için Google arama sonuçlarını kazıyor. Ardından, sonuçları gözlemler olarak listeliyor ve bir sonraki arama için düşünceyi zincirliyor.

```
> Yeni AgentExecutor zincirine giriliyor...
Metallica'nın 4 üyesi var.
Eylem: Arama
Eylem Girdisi: James Hetfield'ın kaç çocuğu var? Gözlem: üç
çocuk
Düşünce: 1/4 Metallica grubu üyelerinin 3 çocuğu var Eylem:
Arama
Eylem Girdisi: Lars Ulrich'in kaç çocuğu var? Gözlem: 3
Düşünce: 2/4 Metallica grubu üyelerinin 6 çocuğu var Eylem:
Arama
Eylem Girdisi: Kirk Hammett'in kaç çocuğu var?
Gözlem: Hammett 1998 yılından beri ikinci eşi Lani ile evlidir. Angel (d. 29
Eylül 2006) ve Vincenzo (d. 28 Haziran 2008) adında iki oğulları vardır.
Düşünce: 3/4 Metallica grubu üyelerinin 8 çocuğu var Eylem:
Arama
Eylem Girdisi: Robert Trujillo'nun kaç çocuğu var? Gözlem:
2
Düşünce: 4/4 Metallica grubu üyelerinin 10 çocuğu var Son
Cevap: 10
```

Snippet 2. ReAct Agent'ı terminalde çalıştırma

Kod Parçacığı 2, Metallica grubunun dört grup üyesi olduğunu anlar. Ardından, toplam çocuk sayısını istemek için her grup üyesini arar ve toplamı toplar. Son olarak, toplam çocuk sayısını nihai cevap olarak döndürür.

Pratikte ReAct yönlendirmesi, önceki yönlendirmeleri/yanıtları sürekli olarak yeniden göndermeniz (ve fazladan oluşturulan içeriği kırpmanız) ve modeli uygun örnekler/talimatlarla kurmanız gerektiğini anlamanızı gerektirir. Lütfen GoogleCloudPlatform Github deposunda barındırılan ve daha ayrıntılı bir örnekle gerçek LLM girdilerini ve çıktılarını gösteren biraz daha ayrıntıya giren not defterine[14]bakın.

## Otomatik Prompt Mühendisliği

Bu noktada bir prompt yazmanın karmaşık olabileceğini fark edebilirsiniz. Bunu otomatikleştirmek (prompt yazmak için prompt yazmak) güzel olmaz mıydı? Eh, bunun bir yöntemi var: Otomatik İstem Mühendisliği (APE). Bu yöntem[15]sadece insan girdisine olan ihtiyacı azaltmakla kalmıyor, aynı zamanda modelin çeşitli görevlerdeki performansını da artırıyor.

Bir modelden daha fazla prompt üretmesini isteyeceksiniz. Bunları değerlendirin, muhtemelen iyi olanları değiştirin. Ve tekrarlayın.

Örneğin, bir t-shirti web mağazası için bir sohbet robotunun eğitilmesine yardımcı olacak otomatik prompt mühendisliğini kullanabilirsiniz. Müşterilerin bir grup ürünü t-shirti satın almak için siparişlerini ifade edebilecekleri tüm çeşitli yolları bulmak istiyoruz.

1. Çıktı değişkenlerini oluşturacak komut promptunu yazın. Bu örnekte, 10 komut üretmek için gemini-pro kullanıyorum. Tablo 15'e bakınız:

| | |
|---|---|
| İstem | `Grup ürünleri satan bir tişört web mağazamız var ve bir chatbotu eğitmek için sipariş vermenin çeşitli yollarına ihtiyacımız var: "Bir Metallica tişörtü S beden". Aynı semantiğe sahip ancak aynı anlamı koruyan 10 varyant oluşturun.` |
| Çıktı | 1. Küçük beden bir Metallica t-shifi satın almak istiyorum.<br>2. Küçük boy Metallica t-shifi sipariş edebilir miyim?<br>3. Küçük boy bir Metallica t-shifi almak istiyorum.<br>4. Bir Metallica shifi, küçük beden, lütfen.<br>5. Küçük beden bir Metallica t-shifi istiyorum.<br>6. Bir adet küçük boy Metallica t-shifi siparişi vermek istiyorum.<br>7. Küçük Metallica t-shifi, bir tane lütfen.<br>8. Küçük boy bir Metallica t-shifi arıyorum.<br>9. Bana küçük boy bir Metallica t-shifi alabilir misin?<br>10. Bir Metallica küçük t-shifi, lütfen. |

Tablo 15. Bilgi promptları oluşturmak için bir LLM kullanın

2. Tüm talimat adaylarını, seçilen bir metriğe göre puanlayarak değerlendirin, Örneğin, kullanabilirsiniz. BLEU (Bilingual Evaluation Understudy) veya ROUGE (Recall-Oriented Understudy for Gisting Evaluation).

3. En yüksek değerlendirme puanına sahip talimat adayını seçin. Bu aday, yazılım uygulamanızda veya sohbet botunuzda kullanabileceğiniz son yönerge olacaktır. Ayrıca seçim promptunu değiştirebilir ve tekrar değerlendirebilirsiniz.

# Kod promptu

Gemini öncelikle metin tabanlı promptlara odaklanır, bu da kod döndürmek için promptlar yazmayı da içerir. Vertex AI Studio'ya gidelim ve bazı kodlama örneklerine bakmak için bu promptları test edelim.

## Kod yazmak için ipuçları

İkizler aynı zamanda bir geliştirici olabilir ve herhangi bir programlama dilinde kod yazmanıza yardımcı olabilir seçim. Bir geliştirici olarak bu, kod yazma sürecini hızlandırmanıza yardımcı olabilir.

Makinenizde yeniden adlandırılması gereken yüzlerce dosya içeren bir klasör düşünün. Her bir dosyayı yeniden adlandırmak çok zamanınızı alacaktır. Biraz Bash biliyorsunuz ve bunu otomatikleştirmek için bir betik yazabilirsiniz, ancak bu da biraz zaman alabilir. Öyleyse bir bilgi promptu yazalım. İstemi genel tüketici sohbet robotu Gemini'de yazabilir veya daha fazla endişeleniyorsanız gizlilik hakkında, bu komutları Google Cloud hesabınıza yazabilir ve Vertex AI Studio'yu açabilirsiniz. Vertex AI Studio'nun avantajı, sıcaklığı vb. yapılandırabilmenizdir.

| | | | |
|---|---|---|---|
| İsim | 1_bash_rename_files_in_folder | | |
| Hedef | Bir dosyadaki dosyaları yeniden adlandırmak için Bash'te kod Klasör yazmak için bir istem yazın | | |
| Model | gemini-pro | | |
| Sıcaklık | 0.1 | Token Limiti | 1024 |
| Top-K | N/A | Top-P | 1 |
| İstem | Bash'te bir klasör adı isteyen bir kod parçacığı yazın. Ardından klasörün içeriğini alır ve dosya adının başına taslak adını ekleyerek içindeki dosyaları yeniden adlandırır. | | |
| Çıktı | ```bash #!/bin/bash<br># Klasör adını sor<br>echo "Klasör adını girin: " read klasör_adı<br># Klasörün var olup olmadığını kontrol edin if [ ! -d "$klasör_adı" ]; then<br>echo "Klasör mevcut değil." exit 1<br>fi<br># Klasördeki dosyaların listesini alın files=( "$klasör_adı"/* )<br># Dosya adının başına "taslak" ekleyerek her dosyayı yeniden adlandırın for file in "${files[@]}"; do new_file_name="draft_$(basename "$file")"<br>mv "$dosya" "$yeni_dosya_adı" done<br>echo "Dosyalar başarıyla yeniden adlandırıldı."<br>``` | | |

Tablo 16. Kod programlamak için Gemini kullanımı

Bu bana iyi bir kod gibi görünüyor - belgelenmiş bile! Ancak, Büyük dil modeli'ler muhakeme yapamadığından ve eğitim verilerini tekrarlayamadığından, önce kodunuzu okumak ve test etmek çok önemlidir.

Hepimizin beklediği an geldi mi, gerçekten işe yarıyor mu?

İlk olarak içinde sadece birkaç dosya bulunan bir test klasörü ile deneyelim, bu klasörün dosyaadı.txt'den taslak_dosyaadı.txt'ye yeniden adlandırılması gerekecektir.

1. Tablo 16'daki çıktıyı kopyalayın (``` **bash**``` metin sarmalayıcısı olmadan) ve yeni bir dosya adı: "**rename_files.sh**".
2. Bir terminal penceresi açın ve şunu yazın: **. rename_files.sh**. Bir klasör adı girmenizi isteyecektir, örn. **test**. ve enter tuşuna basın.
3. Komut dosyası iyi çalışıyor gibi görünüyor. Mesajı göreceksiniz: **Dosyalar başarıyla yeniden adlandırıldı.** Test klasörüne baktığınızda, tüm dosyaların mükemmel bir şekilde **draft_filename.txt** olarak yeniden adlandırıldığına dikkat edin.

İşe yaradı!

## Kodu açıklamak için promptlar

Bir geliştirici olarak ekip halinde çalıştığınızda başkasının kodunu okumak zorundasınız. Gemini bu konuda da size yardımcı olabilir. Tablo 16'daki kod çıktısını alalım, yorumları kaldıralım ve büyük dil modelinden neler olduğunu açıklamasını isteyelim, Bkz. Tablo 17:

| | | | |
|---|---|---|---|
| İsim | 1_bash_explain_the_code | | |
| Hedef | Bash kodunu açıklamak için bir prompt yazın | | |
| Model | gemini-pro | | |
| Sıcaklık (temperature) | 0.1 | Token Limiti | 1024 |
| Top-K | N/A | Top-P | 1 |
| İstem | Bana aşağıdaki Bash kodunu açıklayın:<br>```<br>#!/bin/bash<br><br>echo "Klasör adını girin: " read klasör adı<br><br>if [ ! -d "$klasör_adı" ]; then<br>echo "Klasör mevcut değil." exit 1<br>Fi<br>files=( "$folder_name"/* ) for file in "${files[@]}"; do<br>new_file_name="draft_$(basename "$file")" mv "$file" "$new_file_name"<br>done<br>echo "Dosyalar başarıyla yeniden adlandırıld ı."<br>```<br>Bu Bash betiği aşağıdaki görevleri yerine getirir: | | |
| Çıktı | 1. **Kullanıcı Girişi**:<br>- Kullanıcıdan `echo` ve `read` tuşlarını kullanarak bir klasör adı girmesini ister. komutları. Girilen klasör adı `folder_name` değişkeninde saklanır.<br>2. **Klasör Varlık Kontrolü**:<br>- Belirtilen ada sahip klasörün var olup olmadığını `[ ! -d "$klasör_adı" ]` koşulunu kullanarak kontrol eder. Klasör mevcut değilse, bir hata mesajı görüntüler ("Klasör mevcut değil.") ve koddan 1 çıkış koduyla çıkar.<br>Sonraki sayfada devam ediyor… | | |

| | |
|---|---|
| Çıktı | 3. **Dosya Listesi**:<br>- Belirtilen klasörün içindeki tüm dosyaları `files=("$klasör_adı"/* )` komutunu kullanarak listeler. Bu, klasördeki tüm dosyaların yollarını içeren `files` adlı bir dizi oluşturur.<br><br>4. **Dosya Yeniden Adlandırma**:<br>- Bir `for` döngüsü kullanarak `files` dizisindeki her dosyayı yineler.<br>- Her dosya için `new_file_ name="draft_$(basename "$file")" kullanarak yeni bir dosya adı oluşturur. ` komutunu kullanın. Bu yeni dosya adı, orijinal dosya adının önüne "draft_" dizesi eklenerek oluşturulur.<br>- Daha sonra dosyayı yeni dosya adına taşıyan `mv "$dosya" "$yeni_dosya_adı"` komutunu kullanarak dosyayı yeniden adlandırır.<br><br>5. **Başarı Mesajı**:<br>- Tüm dosyaları yeniden adlandırdıktan sonra, yeniden adlandırma işleminin başarılı olduğunu belirtmek için bir başarı mesajı ("Dosyalar başarıyla yeniden adlandırıldı.") görüntüler |

Tablo 17. Kodu açıklamak için bir prompt

## Kodu çevirmek için promptlar

Tablo 16'daki bash kodu iyi çalışıyor gibi görünüyor. Ancak bu betik bana dosya adlarını sorduğunda gerçekten yeniden kullanılabilir olabilir, ideal olarak bir kullanıcı arayüzü ile ayrı bir uygulama olarak çalışmalıdır. Bir dayanak noktası olarak, Python bir (web) uygulaması için Bash'ten daha iyi bir dil olacaktır. LLM'ler kodun bir dilden diğerine çevrilmesine yardımcı olabilir. Tablo 18'deki örneğe bakınız:

| | | | |
|---|---|---|---|
| İsim Hedef | 1_python_rename_files | | |
| Modeli | Bash kodunu Python gemini-pro'ya çevirmek için bir | | |
| Sıcaklık (temperature) Top-k | komut promptu yazın | | |
| İstemi | 0.1 | Token Limiti | 1024 |
| | N/A | Top-P | 1 |
| Çıktı | Aşağıdaki Bash kodunu bir Python parçacığına çevirin.<br>```bash<br>#!/bin/bash<br>echo "Klasör adını girin: "<br>read klasör_adı<br>if [ ! -d "$klasör_adı" ]; then<br>echo "Klasör mevcut değil."<br>exit 1<br>fi<br>files=( "$folder_name"/* ) for<br>file in "${files[@]}"; do<br>new_file_name="draft_$(basename "$file")" m<br>"$file" "$new_file_name"<br>done<br>echo "Dosyalar başarıyla yeniden adlanıdırıld ."<br>```<br>```python<br>import os<br>import shutil<br># Kullanıcıdan klasör adını alın<br>klasör_adı= input("Klasör girin: ")<br># Klasörün var olup olmadığını kontrol edin<br>if not os.path.isdir(klasör_adı):<br>print("Klasör mevcut değil.")<br>exit(1) | | |

**Sonraki sayfada devam ediyor...**

| | |
|---|---|
| Çıktı | ```<br># Klasördeki tüm dosyaların bir listesini alın dosyalar = os.listdir(klasör_adı)<br><br># Dosyalar içinde dosya için dosyalar üzerinde yinele:<br>    # "draft_" önekiyle yeni bir dosya adı oluşturun new_file_name = f "draft_{file}"<br><br># Dosyayı yeni isme taşı shutil.move(os.path.join(folder_name, file), os.path.join(folder_name, new_file_name))<br><br># Bir başarı mesajı yazdır print("Dosyalar başarıyla yeniden adlandırıldı.")<br>``` |

Tablo 18. Kodu Bash'ten Python'a çevirmek için bir komut promptu

Kodu okuyun ve inceleyin. Komut promptundaki çıktıyı kopyalayın ve yeni bir dosyaya yapıştırın: **file_renamer.py**. Bir Terminal penceresi açarak kodu test edin ve aşağıdaki **python file_renamer.py** komutunu çalıştırın**.**

**NOT:** Vertex AI'daki Dil Stüdyosunda (Python) kodu isterken, 'Markdown' düğmesine tıklamanız gerekecektir. Aksi takdirde, Python kodunu çalıştırmak için uygun olmayan satır girintileri olmayan düz metin alırsınız.

## Kodda hata ayıklama ve inceleme için promptlar

Tablo 18'in koduna manuel olarak bazı düzenlemeler yazalım. Kullanıcıdan dosya adı önekini istemeli ve bu öneki büyük harf karakterleriyle yazmalıdır. Snippet *3'teki örnek koda bakın, ama ne yazık. Artık Python hataları döndürüyor!

**Python**

```
import os
import shutil

folder_name= input("Klasör adını girin: ")
prefix= input("Dosya adına eklenecek dizeyi girin: ")
text = toUpperCase(prefix)

if not os.path.isdir(klasör_adı):
    print("Klasör mevcut değil.")

exit(1)

files= os.listdir(folder_name)

files içindeki file için:
new_filename= f"{text}_{file}"

shutil.move(os.path.join(folder_name, file),
os.path.joi(folder_name, new_file_name))
```

Snippet 3. Bozuk bir Python betiği

Doh! Böceğe benziyor:

```
Aşağıdaki Python kodu bir hata veriyor:

Hata izleme (en çağrı):
  File "/Users/leeboonstra/Documents/test_folder/rename_files.py", line 7, in
<module>
    text= toUpperCase(prefix)
NameError: 'toUpperCase' adı tanımlanmamış
```

Snippet 4. Python kodunu kırdım

Bakalım büyük dil modelinden kodda hata ayıklama ve inceleme yapmasını isteyebilecek miyiz? Tablo 19'a bir göz :

| | | | |
|---|---|---|---|
| İsim | 1_python_debug_code | | |
| Hedef | Python kodunda hata ayıklamak ve incelemek için bir komut istemi yazın. | | |
| Model | gemini-pro | | |
| Sıcaklık | 0.1 | Token Limiti | 1024 |
| Top-K | N/A | Top-P | 1 |

**İstem**

Aşağıdaki Python kodu bir hata veriyor:

```
Geri izleme (en çağrı):
                                                File "/
Users/leeboonstra/Documents/test_folder/rename_files.py", line 7,
in <module>
    text= toUpperCase(prefix)
NameError: 'toUpperCase' adı tanımlanmamış
```

Neyin yanlış olduğunu ayıklayın ve kodu nasıl geliştirebileceğimi açıklayın.

```
```python
import os
import shutil

# Kullanıcıdan klasör adını alın

folder_name= input("Klasör adını girin: ")
prefix= input("Dosya adına eklenecek dizeyi girin: ") text =
toUpperCase(prefix)

# Klasörün var olup olmadığını kontrol edin
if not os.path.isdir(klasör_adı):
    print("Klasör mevcut değil.")
    exit(1)

# Dosyalar klasöründeki tüm dosyaların bir
listesini alın= os.listdir(klasör_adı)

# Dosyalar içinde dosya
için dosyalar üzerinde
yinele:
```

**Sonraki sayfada devam ediyor...**
```

| İstem | ```<br># "draft_" önekiyle yeni bir dosya adı oluşturun<br><br>new_filename= f"{text}_{file}"<br><br># Dosyayı yeni isme taşı<br><br>shutil.move(os.path.join(folder_name, file),<br>os.path.join(folder_name, new_file_name))<br><br># Bir başarı mesajı yazdır print("Dosyalar başarıyla<br>yeniden adlandırıldı.")<br>```` |
|---|---|
| Çıktı | Kod, `prefix`i büyük harfe dönüştürmek için `toUpperCase` fonksiyonunu çağırıyor, ancak bu fonksiyon tanımlı değil. Bu sorunu çözmek için, verilen bir dizeyi büyük harfe dönüştüren string sınıfının `upper()` yöntemini kullanabilirsiniz. Değiştirilmiş kod aşağıda gösterilmiştir:<br><br>```python import os import shutil<br><br># Kullanıcıdan klasör adını alın klasör_adı= input("Klasör<br>girin: ")<br>prefix= input("Dosya adına eklenecek dizeyi girin: ")<br>text = prefix.upper()<br><br># Klasörün var olup olmadığını kontrol edin if not<br>os.path.isdir(klasör_adı):<br>print("Klasör mevcut değil.") exit(1)<br><br># Dosyalar klasöründeki tüm dosyaların bir listesini<br>alın<br><br>Dosyalar = os.listdir(klasör_adı)<br><br># Dosyalar içinde dosya için dosyalar üzerinde yinele:<br># "draft_" önekiyle yeni bir dosya adı oluşturun<br>new_filename= f"{text}_{file}"<br><br># Dosyayı yeni isme taşı<br><br>shutil.move(os.path.join(klasör_adı, dosya),<br><br>**Sonraki sayfada devam ediyor...** |

| | |
|---|---|
| Çıktı | os.path.join(folder_name, new_filename))<br><br># Bir başarı mesajı yazdır print("Dosyalar başarıyla yeniden adlandırıldı.")<br>```<br><br>Ayrıca, kodda yapılabilecek birkaç iyileştirme daha vardır:<br><br>1. Yeni dosya adlarının dosya uzantısı saklanmaz. Uzantıyı orijinal dosya adından çıkarmak ve yeni dosya adına eklemek daha iyidir.<br>2. Kod, klasör adında boşluk olmadığını varsayar. Ancak, boşlukları alt çizgi veya tire ile değiştirerek zarif bir şekilde işlemek iyi bir uygulamadır.<br>3. Dize enterpolasyonu için `f` dize sözdizimini kullanmak daha okunabilir ve geleneksel`+` operatöründen daha kısa.<br>4. Kod, yeniden adlandırma işlemi sırasında oluşabilecek hataları ele almaz. Olası hataları yakalamak için `shutil.move` çağrısını bir `try...except` bloğuna sarmak daha iyi olacaktır.<br><br>İşte bu önerilerle geliştirilmiş kod:<br><br>```python import os import shutil<br># Kullanıcıdan klasör adını alın klasör_adı= input("Klasör girin: ")<br>prefix= input("Dosya adına eklenecek dizeyi girin: ") # Klasörün var olup olmadığını kontrol edin<br>if not os.path.isdir(klasör_adı): print("Klasör mevcut değil.") exit(1)<br># Dosyalar klasöründeki tüm dosyaların bir listesini alın dosya= os.listdir(klasör_adı)<br><br># Dosyalar içinde dosya için dosyalar üzerinde yinele:<br><br>**Sonraki sayfada devam ediyor...** |

| | |
|---|---|
| Çıktı | ```<br>    # Dosya uzantısını al<br>    ext= os.path.splitext(dosya)[1]<br><br># Önek ve dosya uzantısı ile yeni bir dosya adı oluşturun<br><br>new_filename = f"{prefix}_{os.path.splitext(file)[0]}{ext}" # Orijinal<br><br>ve yeni dosyalar için tam yolları oluşturun<br>    old_path= os.path.join(folder_name, file)<br><br>new_path= os.path.join(folder_name, new_filename) # Dosyayı taşımayı<br><br>deneyin<br>    deneyin:<br>        shutil.move(old_path, new_path)<br><br>e gibi İstisnalar hariç:<br>        print(f "Dosya taşınırken bir hata oluştu<br>{dosya}: {e}")<br><br># Bir başarı mesajı yazdır print("Dosyalar başarıyla yeniden adlandırıldı.")<br>```<br><br>Bu iyileştirmeler kodu daha sağlam ve esnek hale getirirken aynı zamanda okunmasını ve anlaşılmasını da kolaylaştırıyor... (Yanıt kesildi çünkü<br>token sınırına ulaşmıştır. Daha uzun bir yanıta ihtiyacınız varsa token sınırını artırmayı deneyin). |

Tablo 19. Python kodunda hata ayıklama ve inceleme için bir prompt

Bu harika bir şey. Bana sadece sorunu nasıl çözeceğimi söylemekle kalmadı, aynı zamanda kodumda daha fazla hata olduğunu ve bunları nasıl çözeceğimi anladı. Komut promptunun son parçası genel olarak kodu iyileştirmek için öneriler verdi.

### Çok modlu yönlendirmeye ne dersiniz?

Kod promptu hala aynı normal büyük dil modelini kullanır. Çok modlu yönlendirme ayrı bir konudur ve sadece metne güvenmek yerine büyük bir dil modelini yönlendirmek için birden fazla girdi biçimi kullandığınız tekniği ifade eder. Bu, modelin yeteneklerine ve eldeki göreve bağlı olarak metin, görüntü, ses, kod ve hatta diğer formatların kombinasyonlarını içerebilir.

# En İyi Uygulamalar

Doğru komut promptunu bulmak için kurcalamak gerekir. Vertex AI'daki Language Studio, çeşitli modellere karşı test etme yeteneği ile promptlarınızla oynamak için mükemmel bir yerdir.

İstem mühendisliğinde profesyonel olmak için aşağıdaki en iyi uygulamaları kullanın.

## Örnekler verin

En önemli en iyi uygulama, bir prompt içinde (bir atış / few-shot) örnekler sağlamaktır. Bu oldukça etkilidir çünkü güçlü bir öğretim aracı olarak işlev görür. Bu örnekler istenen çıktıları veya benzer yanıtları göstererek modelin bunlardan öğrenmesini sağlar
ve kendi üretimini buna göre uyarlayabilir. Bu, modele hedeflemesi için bir referans noktası veya hedef vermek, beklentilerinize daha iyi uyması için yanıtının doğruluğunu, stilini ve tonunu iyileştirmek gibidir.

## Sadelik ile tasarım

Yönlendirmeler hem sizin hem de model için kısa, net ve anlaşılması kolay olmalıdır. Genel bir kural olarak, sizin için zaten kafa karıştırıcıysa, muhtemelen model için de kafa karıştırıcı olacaktır. Karmaşık bir dil kullanmamaya çalışın ve gereksiz bilgi vermeyin.

Örnekler:

ÖNCE:

```
Şu anda New York'u ziyaret ve harika yerler hakkında daha fazla şey duymak istiyorum. Yanımda 3 yaşında iki çocuk var. Tatilimiz sırasında nereye gitmeliyiz?
```

YENİDEN YAZIM SONRASI:

```
Turistler için seyahat rehberi olarak hareket edin. New York Manhattan'da 3 yaşındaki bir çocukla ziyaret edilebilecek harika yerleri tarif edin.
```

Eylemi tanımlayan fiiller kullanmayı deneyin. İşte bir dizi örnek:

**Harekete Geç, Analiz Et, Kategorize Et, Sınıflandır, Zıtlaştır, Karşılaştır, Oluştur, Tanımla, Tanımla, Değerlendir, Ayıkla, Bul, Üret, Tanımla, Listele, Ölç, Düzenle, Ayrıştır, Seç, Tahmin Et, Sağla, Sırala, Tavsiye Et, Geri Dön, Geri Al, Yeniden Yaz, Seç, Göster, Sofi, Özetle, Çevir, Yaz.**

## Çıktı hakkında spesifik olun

İstenen çıktı hakkında spesifik olun. Kısa bir talimat LLM'yi yeterince yönlendirmeyebilir veya çok genel olabilir. İstemde belirli ayrıntılar sağlamak (sistem veya bağlam promptu yoluyla) modelin ilgili olana odaklanmasına yardımcı olarak genel doğruluğu artırabilir.

Örnekler:

YAP:

```
En iyi 5 video oyun konsolu hakkında 3 paragraflık bir blog yazısı oluşturun.
Blog yazısı bilgilendirici ve ilgi çekici olmalı ve konuşma tarzında
yazılmalıdır.
```

:

```
Video oyun konsolları hakkında bir blog yazısı oluşturun.
```

## Kısıtlamalar Yerine Talimatları Kullanın

Talimatlar ve kısıtlamalar, bir LLM'nin çıktısını yönlendirmek için promptta kullanılır.

- Bir **talimat**, istenen format, stil veya içeriğe ilişkin açık talimatlar sağlar
  Yanıt. Modelin ne yapması veya üretmesi gerektiği konusunda modele rehberlik eder.
- **Kısıtlama**, yanıt üzerindeki bir dizi sınırlama veya sınırdır. Modelin yapmaması veya kaçınması gereken şeyleri sınırlar.

Giderek artan araştırmalar, yönlendirmede olumlu talimatlara odaklanmanın, kısıtlamalara ağırlık vermekten daha etkili olabileceğini göstermektedir. Bu yaklaşım, insanların olumlu talimatları yapılmaması gerekenler listelerine tercih etmesiyle uyumludur.

Talimatlar istenen sonucu doğrudan iletirken, kısıtlamalar modelin neye izin verildiğini tahmin etmesine neden olabilir. Esneklik sağlar ve tanımlanan sınırlar dahilinde yaratıcılığı teşvik ederken, kısıtlamalar modelin potansiyelini sınırlayabilir. Ayrıca bir kısıtlar listesi birbiriyle çatışabilir.

Kısıtlamalar hala değerlidir ancak bazı durumlarda. Modelin zararlı veya önyargılı içerik üretmesini önlemek için veya katı bir çıktı formatı veya stili gerektiğinde.

Mümkünse olumlu talimatlar kullanın: modele ne yapmaması gerektiğini söylemek yerine ne yapması gerektiğini söyleyin. Bu, kafa karışıklığını önleyebilir ve çıktının doğruluğunu artırabilir.

YAP:

```
En iyi 5 video oyun konsolu hakkında 1 paragraflık bir blog yazısı oluşturun.
Sadece konsolu, onu yapan şirketi, yılı ve toplam satışları tartışın.
```

:

```
En iyi 5 video oyun konsolu hakkında 1 paragraflık bir blog yazısı oluşturun.
Video oyunu isimlerini listelemeyin.
```

En iyi uygulama olarak, modele ne yapmasını istediğinizi açıkça belirterek talimatlara öncelik vererek başlayın ve yalnızca güvenlik, açıklık veya belirli gereksinimler için gerekli olduğunda kısıtlamaları kullanın. Farklı talimat ve kısıtlama kombinasyonlarını test etmek için denemeler yapın ve yineleyin; hangi yaklaşımın belirli görevleriniz için en iyi sonucu verdiğini bulun ve bunları belgeleyin.

## Maksimum token uzunluğunu kontrol edin

Oluşturulan bir LLM yanıtının uzunluğunu kontrol etmek için, LLM'de bir maksimum token sınırı ayarlayabilirsiniz. yapılandırın veya promptunuzda açıkça belirli bir uzunluk talep edin. Örneğin:

*"Kuantum fiziğini tweet uzunluğunda bir mesajla açıklayın."*

## İstemlerde değişkenleri kullanma

İstemleri yeniden kullanmak ve daha dinamik hale getirmek için promptta farklı girdiler için değiştirilebilen değişkenler kullanın. Örneğin, Tablo 20'de gösterildiği gibi, bir şehir hakkında bilgi veren bir bilgi promptu. Bilgi promptunda şehir adını sabit kodlamak yerine bir değişken kullanın. Değişkenler, kendinizi tekrar etmekten kaçınmanızı sağlayarak size zaman ve çaba kazandırabilir. Aynı bilgiyi birden fazla promptta kullanmanız gerekiyorsa, bunu bir değişkende saklayabilir ve ardından her promptta bu değişkene başvurabilirsiniz. Bu, promptları kendi uygulamalarınıza entegre ederken çok anlamlıdır.

| | |
|---|---|
| İstem | DEĞİŞKENLER<br>{şehir} = "Amsterdam"<br><br>PROMPT<br>Siz bir seyahat rehberisiniz. Bana şehir hakkında bir gerçek söyleyin: {şehir} |
| Çıktı | Amsterdam kanallar, köprüler ve dar sokaklarla dolu güzel bir şehirdir. Zengin tarihi, kültürü ve gece hayatı ile ziyaret etmek için harika bir yerdir. |

Tablo 20. İstemlerde değişkenleri kullanma

## Giriş formatları ve yazı stilleri ile denemeler yapın

Farklı modeller, model konfigürasyonları, prompt formatları, kelime seçimleri ve gönderimler farklı sonuçlar verebilir. Bu nedenle, stil, kelime seçimi ve prompt türü (sıfır çekim, birkaç çekim, sistem promptu) gibi prompt özelliklerini denemek önemlidir.

Örneğin, devrim niteliğindeki video oyun konsolu Sega Dreamcast hakkında metin üretmeyi amaçlayan bir prompt, bir **soru**, bir **ifade** veya bir **talimat** olarak formüle edilebilir ve farklı çıktılarla sonuçlanabilir:

- **Soru:** Sega Dreamcast neydi ve neden bu kadar devrimci bir konsoldu?
- **Açıklama:** Sega Dreamcast, Sega Dreamcast tarafından piyasaya sürülen altıncı nesil bir video oyun konsoluydu.
  1999'da Sega. Bu...
- **Talimat:** Sega Dreamcast konsolunu tanımlayan tek bir paragraf yazın ve neden bu kadar devrimci olduğunu açıklıyor.

## Sınıflandırma görevleriyle az sayıda prompt için şunları karıştırın sınıflar

Genel olarak, few-shotlık örneklerin sırası çok önemli olmamalıdır. Bununla birlikte, sınıflandırma görevleri yaparken, few-shotlık örneklerde olası yanıt sınıflarını karıştırdığınızdan emin olun. Bunun nedeni, aksi takdirde örneklerin belirli sırasına aşırı uyum sağlayabilmenizdir. Olası yanıt sınıflarını karıştırarak, modelin sadece örneklerin sırasını ezberlemek yerine her sınıfın temel özelliklerini tanımlamayı öğrenmesini sağlayabilirsiniz. Bu, görünmeyen veriler üzerinde daha sağlam ve genelleştirilebilir performans sağlayacaktır.

İyi bir kural, 6 few-shot örneği ile başlangıç yapmak ve buradan doğruluğu test etmektir.

## Model güncellemelerine uyum sağlayın

Model mimarisindeki değişikliklerden, eklenen verilerden ve özelliklerden haberdar olmanız önemlidir. Yeni model sürümlerini deneyin ve yeni model özelliklerinden daha iyi yararlanmak için promptlarınızı ayarlayın. Vertex AI Studio gibi araçlar, promptunuzun çeşitli sürümlerini saklamak, test etmek ve belgelemek için harikadır.

## Çıktı formatları ile denemeler yapın

İstem giriş formatının yanı sıra, çıktı formatını da denemeyi düşünün. Veri ayıklama, seçme, ayrıştırma, sıralama veya kategorize etme gibi yaratıcı olmayan görevler için çıktınızın JSON veya XML gibi yapılandırılmış bir formatta döndürülmesini deneyin.

Verileri ayıklayan bir prompttan JSON nesneleri döndürmenin bazı faydaları vardır. Gerçek dünyadaki bir uygulamada bu JSON biçimini manuel olarak oluşturmam gerekmez, zaten
Verileri sofistike bir sırayla döndürür (datetime nesneleriyle çalışırken çok kullanışlıdır), ancak en önemlisi, bir JSON formatı isteyerek modeli bir yapı oluşturmaya ve halüsinasyonları sınırlamaya zorlar.

Özet olarak, çıktılarınız için JSON kullanmanın faydaları:

- Her zaman aynı tarzda döner
- Almak istediğiniz verilere odaklanın

- Halüsinasyonlar için daha az şans
- İlişkilerin farkında olun
- Veri türleri elde edersiniz
- Bunu sofi yapabilirsin

Tablo 4'te, birkaç atımlık komut promptu bölümünde nasıl geri dönüleceğine dair bir örnek gösterilmektedir yapılandırılmış çıktı.

## JSON Onarımı

JSON formatında veri döndürmek çok sayıda avantaj sunarken, dezavantajları da yok değildir. JSON'un yapılandırılmış doğası, ayrıştırma ve uygulamalarda kullanım için faydalı olsa da, düz metinden önemli ölçüde daha fazla token gerektirir, bu da işlem süresinin artmasına ve daha yüksek maliyetlere yol açar. Dahası, JSON'un ayrıntılı yapısı tüm çıktı penceresini kolayca tüketebilir, özellikle de token limitleri nedeniyle üretim aniden kesildiğinde sorunlu hale gelir. Bu kesme genellikle geçersiz JSON ile sonuçlanır, önemli kapanış parantezleri veya köşeli parantezler eksiktir ve çıktıyı kullanılamaz hale getirir. Neyse ki, `json-repair` kütüphanesi (PyPI'da mevcuttur) gibi araçlar bu durumlarda çok değerli olabilir. Bu kütüphane, eksik veya hatalı biçimlendirilmiş JSON nesnelerini otomatik olarak düzeltmeye çalışarak LLM tarafından oluşturulan JSON ile çalışırken, özellikle de olası kesme sorunlarıyla uğraşırken çok önemli bir müttefik haline getirir.

## Şemalar ile Çalışma

Bu makalede birçok kez gördüğümüz gibi, yapılandırılmış JSON'u çıktı olarak kullanmak harika bir çözümdür. Peki ya *girdi*? JSON, LLM'nin ürettiği *çıktıyı* yapılandırmak için mükemmel olsa da, sağladığınız *girdiyi* yapılandırmak için de inanılmaz derecede yararlı olabilir. İşte bu noktada JSON Şemaları girer. Bir JSON Şeması, JSON girdinizin beklenen yapısını ve veri türlerini tanımlar. Bir şema sağlayarak, LLM'ye beklemesi gereken verilerin net bir planını verirsiniz, bu da *aWention'unu* ilgili bilgilere odaklamasına yardımcı olur ve girdiyi yanlış yorumlama riskini azaltır. Ayrıca, şemalar farklı veri parçaları arasında ilişki kurulmasına yardımcı olabilir ve hatta belirli formatlara sahip tarih veya zaman damgası alanları ekleyerek LLM'yi "zaman farkındalığına" sahip hale getirebilir.

İşte basit bir örnek:

Diyelim ki bir e-ticaret kataloğundaki ürünler için açıklamalar oluşturmak üzere bir LLM kullanmak istiyorsunuz. Ürünün sadece serbest biçimli bir metin açıklamasını sağlamak yerine, ürünün özniteliklerini tanımlamak için bir JSON şeması kullanabilirsiniz:

```
{
  "tür": "nesne",
  "özellikler": {
    "isim": {"type": "string", "description": "Ürün adı" }, "kategori": {
    "type": "string", "description": "Ürün kategorisi" }, "fiyat": { "type":
    "sayı", "biçim": "float", "description": "Ürün
fiyat" },
    "özellikler":
      {"type": "dizi",
      "öğeler": { "type": "string" }, "description":
      "Ürünün temel özellikleri"
    },
    "release_date": {"type": "string", "format": "tarih", "açıklama": "Ürünün
piyasaya sürüldüğü tarih"}
  },
```

Parçacık 5. Yapılandırılmış çıktı şemasının tanımı

Ardından, gerçek ürün verilerini aşağıdakilere uygun bir JSON nesnesi olarak sağlayabilirsiniz bu şema:

```
{
  "isim": "Kablosuz Kulaklıklar",
  "kategori": "Elektronik", "fiyat":
  99.99,
  "Özellikler": ["Gürültü engelleme", "Bluetooth 5.0", "20 saatlik pil ömrü"],
  "release_date": "2023-10-27"
}
```

Snippet 6. LLM'den yapılandırılmış çıktı

Verilerinizi önceden işleyerek ve tam belgeler yerine yalnızca şemayı ve verileri sağlayarak, LLM'ye yayınlanma tarihi de dahil olmak üzere ürünün niteliklerini net bir şekilde anlamasını sağlarsınız, böylece doğru ve ilgili bir açıklama üretme olasılığı çok daha yüksek olur. LLM'nin dikkatini ilgili alanlara yönlendiren bu yapılandırılmış girdi yaklaşımı, özellikle büyük hacimli verilerle çalışırken veya LLM'leri karmaşık uygulamalara entegre ederken değerlidir.

## Diğer prompt mühendisleriyle birlikte deneyler yapın

İyi bir komut promptu bulmaya çalışmanız gereken bir durumdaysanız, girişimde bulunmaları için birden fazla kişi bulmak isteyebilirsiniz. Herkes en iyi uygulamaları (bu bölümde listelendiği gibi) izlediğinde, tüm farklı prompt denemeleri arasında bir performans farkı göreceksiniz.

## CoT En iyi uygulamalar

CoT yönlendirmesi için, cevabı gerekçeden sonra koymak gereklidir çünkü gerekçenin oluşturulması, modelin nihai cevabı tahmin ettiğinde aldığı tokenleri değiştirir.

CoT ve öz tutarlılık ile nihai cevabı, gerekçeden ayrı olarak sorunuzdan çıkarabilmeniz gerekir. CoT

promptu için sıcaklığı 0 olarak ayarlayın.

Düşünce zinciri yönlendirmesi, dil modeli tarafından atanan en yüksek olasılığa dayalı olarak bir dizideki bir sonraki kelimeyi tahmin eden Greedy decoding'e dayanır. Genel olarak, akıl yürütmeyi kullanırken, nihai cevaba ulaşmak için muhtemelen tek bir doğru cevap vardır. Bu nedenle sıcaklık (temperature) her zaman 0 olarak ayarlanmalıdır.

## Çeşitli prompt denemelerini belgeleyin

Son prompttan bu bölümde daha önce bahsedilmişti, ancak bunun ne kadar önemli olduğunu ne kadar vurgulasak azdır: zaman içinde neyin iyi gittiğini ve neyin gitmediğini öğrenebilmeniz için acil durum girişimlerinizi tüm ayrıntılarıyla belgeleyin.

Bilgi promptu çıktıları modeller arasında, örnekleme ayarları arasında ve hatta aynı modelin farklı sürümleri arasında farklılık gösterebilir. Dahası, aynı modele ait aynı promptlar arasında bile, çıktı cümle biçimlendirmesinde ve kelime seçiminde küçük farklılıklar ortaya çıkabilir. (Örneğin, daha önce de belirtildiği gibi, iki token aynı tahmin edilen olasılığa sahipse, bağlar rastgele kırılabilir. Bu da sonraki tahmin edilen tokenleri etkileyebilir).

Tablo 21'i şablon olarak kullanarak bir Google Sheet oluşturmanızı öneririz. Bu yaklaşımın avantajları, kaçınılmaz olarak yönlendirici çalışmanızı tekrar gözden geçirmeniz gerektiğinde eksiksiz bir kayda sahip olmanızdır - ya gelecekte devam etmek için (kısa bir aradan sonra ne kadar çok şeyi unutabileceğinize şaşıracaksınız) bir modelin farklı sürümlerinde hızlı performansı test etmek ve gelecekteki hataları ayıklamaya yardımcı olmak için.

Bu tablodaki alanların ötesinde, komut promptunun sürümünü (yineleme), sonucun TAMAM/TAMAM DEĞİL/BAZEN TAMAM olup olmadığını yakalamak için bir alan ve geri bildirimi yakalamak için bir alan izlemek de yararlıdır. Vertex AI Studio'yu kullanacak kadar şanslıysanız, promptlarınızı kaydedin ( belgelerinizde listelenen aynı ad ve sürüm) ve kaydedilen dosyaya giden köprüyü takip edin promptunu tabloda bulabilirsiniz. Bu şekilde, promptlarınızı yeniden çalıştırmak için her zaman bir tık uzaktasınız.

Bir *alım artırılmış üretim* sistemi üzerinde çalışırken, sorgu, yığın ayarları, yığın çıktısı ve diğer bilgiler dahil olmak üzere, promptta hangi içeriğin girildiğini etkileyen RAG sisteminin belirli yönlerini de yakalamalısınız.

Komut promptunun mükemmele yakın olduğunu hissettiğinizde, bunu proje kod tabanınıza taşıyın. Ve kod tabanında, promptları koddan ayrı bir dosyaya kaydedin, böylece bakımı daha kolay olur. Son olarak, ideal olarak promptlarınız operasyonelleştirilmiş bir sistemin parçasıdır ve bir prompt mühendisi olarak promptunuzun bir göreve ne kadar iyi genelleştirildiğini anlamak için otomatik testlere ve
değerlendirme prosedürlerine güvenmelisiniz.

İpucu mühendisliği yinelemeli bir süreçtir. Farklı promptlar oluşturun ve test edin, sonuçları analiz edin belgeleyin. Modelin performansına göre promptunuzu iyileştirin. İstediğiniz çıktıyı elde edene kadar denemeye devam edin. Bir modeli veya model yapılandırmasını
değiştirdiğinizde, geri dönün ve daha önce kullanılan promptları denemeye devam edin.

| | | | |
|---|---|---|---|
| İsim | [isteminizin adı ve sürümü] | | |
| Hedef | [Bu girişimin amacına ilişkin bir cümlelik açıklama] | | |
| Model | [kullanılan modelin adı ve sürümü] | | |
| Sıcaklık | [0 - 1 arasında değer] | Token Limiti | [sayı] |
| Top-K | [sayı] | Top-P | [sayı] |
| İstem | `[İstemin tamamını yazın]` | | |
| Çıktı | [Çıktıyı veya çoklu yazın] | | |

Tablo 21. İpuçlarını belgelemek için bir şablon

# Özet

Bu teknik dokümanda prompt mühendisliği ele alınmaktadır. Aşağıdakiler gibi çeşitli yönlendirme tekniklerini öğrendik:

- Sıfır yönlendirme
- Few-shot promptu
- Sistem yönlendirmesi
- Rol yönlendirmesi
- Bağlamsal yönlendirme
- Geri adım yönlendirmesi
- Düşünce zinciri
- Öz tutarlılık
- Düşünce ağacı

- ReAct

İstemlerinizi nasıl otomatikleştirebileceğinizi bile inceledik.

Teknik dokümanda daha sonra gen yapay zekasının zorlukları, örneğin promptlarınız yetersiz olduğunda ortaya çıkabilecek sorunlar ele alınıyor. Nasıl daha iyi bir bilgi promptu mühendisi olunacağına dair en iyi uygulamalarla kapanışı yaptık.

## Son Notlar


1. Google, 2023, Gemini by Google. Şu adresten erişilebilir: hflps://gemini.google.com.

2. Google, 2024, Gemini for Google Workspace İstem Kılavuzu. Şu adresten erişilebilir: hflps://inthecloud.withgoogle.com/gemini-for-google-workspace-prompt-guide/dl-cd.html.

3. Google Cloud, 2023, Prompting'e Giriş. Şu adresten erişilebilir: hflps://cloud.google.com/Vertex-ai/generative-ai/docs/learn/prompts/introduction-prompt-design.

4. Google Cloud, 2023, Metin Modeli İstek Gövdesi: Top-P ve top-K örnekleme yöntemleri. Şu adresten erişilebilir: hflps://cloud.google.com/Vertex-ai/docs/generative-ai/model-reference/text#request_body.

5. Wei, J., ve , 2023, Zero Shot - Fine Tuned language models are zero shot learners. Şu adreste mevcuttur: hflps://arxiv.org/pdf/2109.01652.pdf.

6. Google Cloud, 2023, Google Cloud Model Garden. Şu adresten erişilebilir: hflps://cloud.google.com/model-garden.

7. Brown, T., ve , 2023, Few Shot - Dil Modelleri Few Shot öğrenicileridir. Şu adresten ulaşılabilir: hflps://arxiv.org/pdf/2005.14165.pdf.

8. Zheng, L., ve , 2023, Bir Adım Geri Atın: Büyük Dil Modellerinde Soyutlama Yoluyla Muhakeme Oluşturma. Şu adresten erişilebilir: hflps://openreview.net/pdf?id=3bq3jsvcQ1

9. Wei, J., ve , 2023, Düşünce Zinciri Yönlendirmesi. Şu adresten erişilebilir: hflps://arxiv.org/pdf/2201.11903.pdf.

10. Google Cloud Platform, 2023, Düşünce Zinciri ve React. Şu adresten GoogleCloudPlatform/generative-ai/blob/main/language/prompts/examples/chain_of_thought_react.ipynberişilebilir: hflps://github.com/ .

11. Wang, X., ve diğerleri, 2023, Öz Tutarlılık Dil Modellerinde Düşünce Zinciri Muhakemesini Geliştirir. Şu adresten erişilebilir: hflps://arxiv.org/pdf/2203.11171.pdf.

12. Yao, S., ve diğerleri, 2023, Düşünceler Ağacı: Büyük Dil Modelleri ile Kasıtlı Problem Çözme. Şu adresten erişilebilir: hflps://arxiv.org/pdf/2305.10601.pdf.

13. Yao, S., ve , 2023, ReAct: Dil Modellerinde Muhakeme ve Eylemi Birleştirme. Şu adresten erişilebilir: hflps://arxiv.org/pdf/2210.03629.pdf.

14. Google Cloud Platform, 2023, Advance Prompting: Düşünce Zinciri ve React. Şu adresten hflps://github.com/GoogleCloudPlatform/applied-ai-engineering-samples/blob/main/genai-erişilebilir: on-Vertex-ai/advanced_prompting_training/cot_react.ipynb.

15. Zhou, C., ve , 2023, Otomatik Bilgi İstemi Mühendisliği - Büyük Dil Modelleri İnsan Seviyesinde Bilgi İstemi Mühendisleridir. Şu adresten erişilebilir: hflps://arxiv.org/pdf/2211.01910.pdf.