**Daha iyi RAG için yöntemler:** <https://www.linkedin.com/pulse/rag-notes-building-better-rag-part-1-akshay-chougule-ldr4e/>

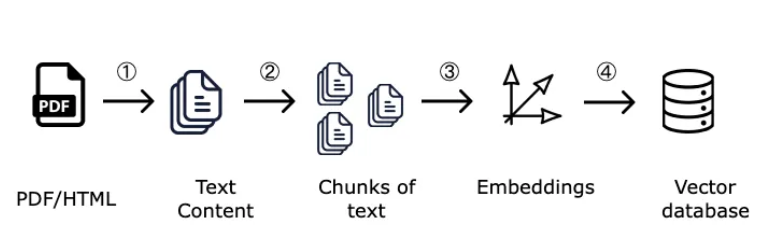
**CONTEXT AWARE CHUNKING FOR RAG (RAG İÇİN BAĞLAMA DUYARLI PARÇALAMA)**

* <https://medium.com/@glorat/context-aware-chunking-for-enhanced-retrieval-augmented-generation-oct23-9dcd435d9cf1>

***unstructured.io*** tarafından sağlanan bağlama duyarlı parçalamanın entegrasyonu

***token:*** metnin küçük parçalarını veya dil öğelerini temsil eder. LLM (Large Language Model) modellerindeki token sayısı, genellikle modelin boyutunu, karmaşıklığını ve performansını belirler.

Belgeden Vector DB’ye Standart Adımlar



* DocumentLoader: örneğin UnstructuredLoader
* TextSplitter: örneğin RecursiveCharacterTextSplitter
* Embeddings: örneğin OpenAIEmbeddings
* VectorStore: örneğin VectorStore

**RAG Örneği:**

"Bana yolculuklarla ilgili ipuçları verin.” sorusunu sorduk.

Öncelikle belgenin düz metne dönüştürülmesi gerekir. DocumentLoader bunun gibi metinler üretir. Ardından parçalama başlar. Ama TextSplitter sonuçları istenilmeyen şekilde 2 chunk a bölebilir.

Şimdi soru en iyi 1. chunk tarafından yanıtlanır çünkü bu yığın yolculuklar kelimesini içerir - ancak sonuçta ortaya çıkan LLM sorusu "bilmiyorum" diyecektir çünkü bağlam aslında yolculuk ipuçlarını içermez. Cevabı yanlış verecek. Yolculuklarla ilgili anlamsal bilgi yığını bölünmüştür, bu nedenle bilgi yığın içinde kaybolmuştur ve bir araya getirilemez.

Bu sorun normalde TextSplitter'ın ***"chunkOverlap"*** anahtar parametresiyle yapılandırılmasıyla çözülür. Bu, her bir parça boyunca bilgide bir miktar örtüşme olmasını sağlayarak, bir parça içindeki bağlamın bozulma olasılığını azaltır.

Artık Soru-Cevap alıcısı 2. chunk ı en kullanışlı olarak seçiyor, gerekli tüm içerik parçanın içinde. Ama LLM cevabı yine yanlış veriyor. Buradaki sorun muhtemelen LLM'nin hatası değil, **parçalamanın bir sonucu** olarak zayıf veri kalitesidir.

Chunklar 3 e bölünerek her chunk ayrı konuları ele alacak şekilde oluşturulabilir. Çünkü her bir parça semantik olarak ilgili bağlamsal bilgileri örtüşmeden düzgün bir şekilde gruplandırılmış olur.

Keşke isteğe bağlı belge türlerini (PDF/HTML/vb.) işleyebilen, aynı zamanda bunları yorumlayıp başlığa göre gruplandırabilen bir teknoloji olsaydı.

[**unstructured/0.10.15'ten**](https://github.com/Unstructured-IO/unstructured/releases/tag/0.10.15) **bu yana, belgeleri doğrudan doğru boyutta parçalara dönüştürerek tam olarak gerektiği gibi bölmek için chunking\_strategy=by\_title parametresini destekleyen unstructured.io'ya girin. Ve** [**unstructured-api/.0.47**](https://github.com/Unstructured-IO/unstructured-api/releases/tag/0.0.47) **ayrıca API'leri aracılığıyla parçaları istenilen gibi geri almayı da destekler.**

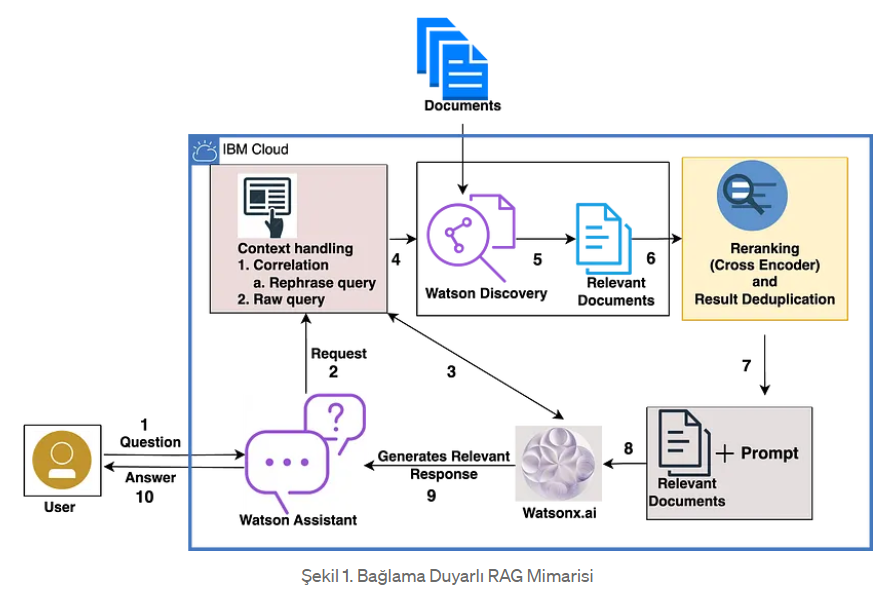
***chunking\_strategy=by\_title*** parametresi, belgelerin doğrudan doğruya başlıklara göre bölünmesini sağlar.

* <https://unstructured.io/blog/optimizing-unstructured-data-retrieval>

RAG sistemlerinde Yapılandırılmamış metin ve meta veri çıkarmanın Çam Kozalağı Hibrit Arama ile birleştirilmesi, benzersiz bir veri depolama ve alma yaklaşımı sunar. Bu kombinasyon, alınan bilgilerin doğruluğunu ve uygunluğunu artırır ve süreci düzene sokarak daha verimli ve kullanıcı dostu hale getirir.

* <https://pypi.org/project/context-aware-chunker/> : kütüphane
* <https://www.datarobot.com/ai-accelerators/using-the-jitr-bot-to-generate-context-aware-responses/> : bir uygulama
* <https://www.youtube.com/watch?v=Rt6G3QaEAUA> : video, chunking yöntemleri
* <https://medium.com/@pinkal08cece/context-aware-retrieval-augmented-generation-rag-b9713a32c858>

**ÖNERİLEN YÖNTEM:**



**Adım 1:** Kullanıcı, Watson Assistant'a bir soru sorar.

**Adım 2 ve 3:** Watson Assistant, iki istem şablonunu yürütmek için ***Watsonx.ai'yı*** kullanarak isteği ***"Context Handling”*** (Bağlam İşleme) Katmanına iletir. İlk şablon, Flan-t5-xl modeli kullanılarak mevcut soru ile önceki soru-cevaplar arasındaki korelasyonu değerlendirmek için hazırlanmıştır; ikinci şablon ise, Llama-2–70B modeli kullanılarak bir korelasyon belirlenirse mevcut sorunun yeniden ifade edilmesini içerir.

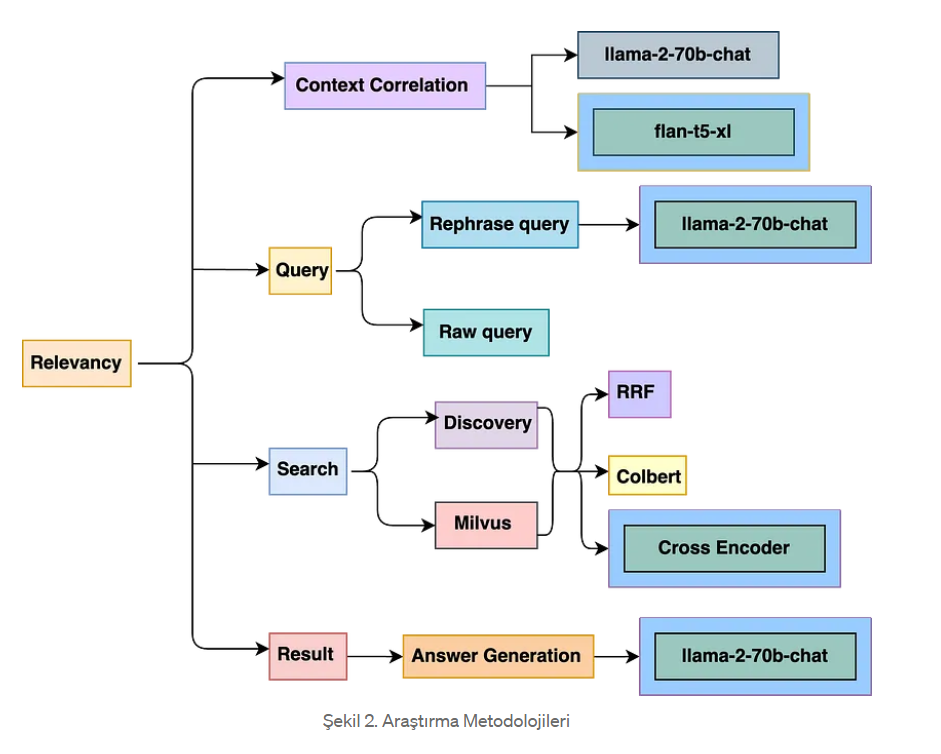
**Adım 4 ve 5:** İlgili belgeler için Watson Discovery'yi sorgulamak amacıyla ham ve yeniden ifade edilen sorgular kullanılır.

**Adım 6:** Çapraz kodlayıcı yeniden sıralama (ms-marco-MiniLM-L-6-v2 modeli) ve tekilleştirme teknikleri, alınan belgelerdeki ilgili pasajların yerini belirlemek için uygulanır.

**Adım 7:** İlgili pasajlar ve kullanıcının sorusu kullanılarak bir bilgi istemi şablonu oluşturulur.

**Adım 8:** Watsonx.ai, Llama-2–70B modelini kullanarak bir yanıt oluşturma istemiyle çağrılır.

**Adım 9 ve 10:** Oluşturulan yanıt Watson Assistant'a gönderilir ve kullanıcıya görüntülenir.



**Correlaction:** Korelasyon değerlendirmesinde, korelasyonu değerlendirmek için ***LLama-70B*** ve ***Flan-t5*** modelleri ile deneyler yapılmıştır. Bulgular, Flan-t5'in yaklaşımımızda dikkate alınan belirli veri kümesi için korelasyon sınıflandırmasında *LLama-70B'den daha iyi performans* gösterdiğini göstermektedir. Değerlendirme için insan değerlendirmesini kullanılmıştır.

**Query:** Bir sorgu yöntemi üzerinde üç farklı yaklaşım denenmiştir. Birincisi ham sorgu, ikincisi hibrit (ham + yeniden ifade edilmiş sorgu), üçüncüsü ise sorguyu her seferinde yeniden ifade etmek. Hibrit diğerlerine kıyasla daha iyi çalışmaktadır. Not: Rephrase sorgusu LLama-70B modeli kullanılarak yürütülmüştür.

**Search:** Arama sorgusundaki ilgili pasajların alınmasını optimize etmek için ***Watson Discovery (****doğal dil işleme (NLP) yeteneklerini kullanarak belgeleri ve metin verilerini analiz eden bir servis****)*** ve ***Milvus******(****vektör tabanlı arama için optimize edilmiş açık kaynaklı bir veri tabanı****)*** gibi veri tabanlarını kullanıldı. En etkili ayarları belirlemek için hem Discovery hem de Milvus üzerinde çeşitli konfigürasyonlarla çok sayıda deneme yapılmıştır. Ek olarak, en uygun pasajları elde etmek için ***RRF (Reciprocal Rank Fusion) (****birden fazla arama motorundan gelen sonuçları birleştirerek daha iyi sonuçlar elde etmeyi amaçlayan bir yöntem****)***, ***ColBERT (****belge arama modelleri için özel olarak tasarlanmış bir dizi dikkat mekanizması olan bir model****)*** ve ***Cross-Encoder (****iki farklı metni alır ve bunların arasındaki ilişkiyi belirler****)*** dahil olmak üzere reranker teknikleri uygulanmıştır. Özellikle, ***ms-marco-MiniLM-L-6-v2 modeline sahip Cross-Encoder***, diğer yapılandırmalara kıyasla ***daha******üstün sonuçlar*** vermiştir.

**Result:** Yanıtları oluşturmak için, Llama-2-70B modelini çağırmak üzere Watsonx.AI kullanılmıştır.

* <https://medium.com/@kenzic/getting-started-chunking-strategy-ebd4ab81f745>

Context aware chunking, daha kesin bir parçalama stratejisi uygulamak için bağlamın doğasını kullanan bir grup parçalama yöntemidir.

Python uygulaması yerleşik cümle bölme özelliğiyle birlikte gelir. Ama *langchain.js* henüz bu özelliğe sahip değil. Ancak özel cümle ayırıcı kolaylıkla uygulanabilir. Ayırcıyı oluşturmak için 'TextSplitter' sınıfını genişletilebilir.



Cümle bölme işleminde daha sağlam bir uygulama için : <https://naturalnode.github.io/natural/> kütüphaneleri kullanılabilir.

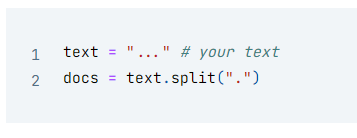
* <https://www.pinecone.io/learn/chunking-strategies/>

Context aware chunking, parçalanan içeriğin doğasından yararlanmak ve ona daha karmaşık parçalama uygulamak için kullanılan bir dizi yöntemdir.

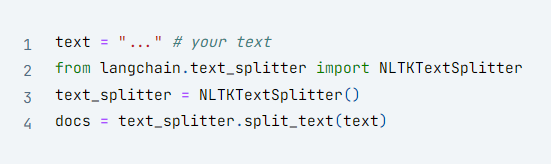
**Cümle Bölme**

Birçok model, cümle düzeyinde içerik yerleştirmek için optimize edilmiştir. Doğal olarak cümle parçalamayı kullanırız ve bunu yapmak için aşağıdakiler de dahil olmak üzere çeşitli yaklaşımlar ve araçlar mevcuttur:

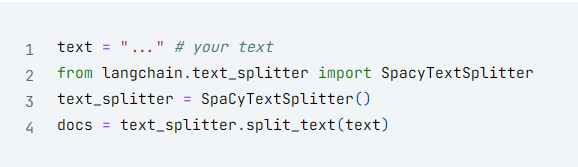
* **Naive splitting:** Cümleleri noktalarla (".") ve yeni satırlarla bölmektir. Bu hızlı ve basit olsa da, bu yaklaşım tüm olası uç durumları dikkate almaz.



* **NLTK:** İnsan dili verileriyle çalışmak için popüler bir Python kütüphanesidir. Metni cümlelere bölebilen ve daha anlamlı parçalar oluşturmaya yardımcı olan bir cümle belirteci sağlar.



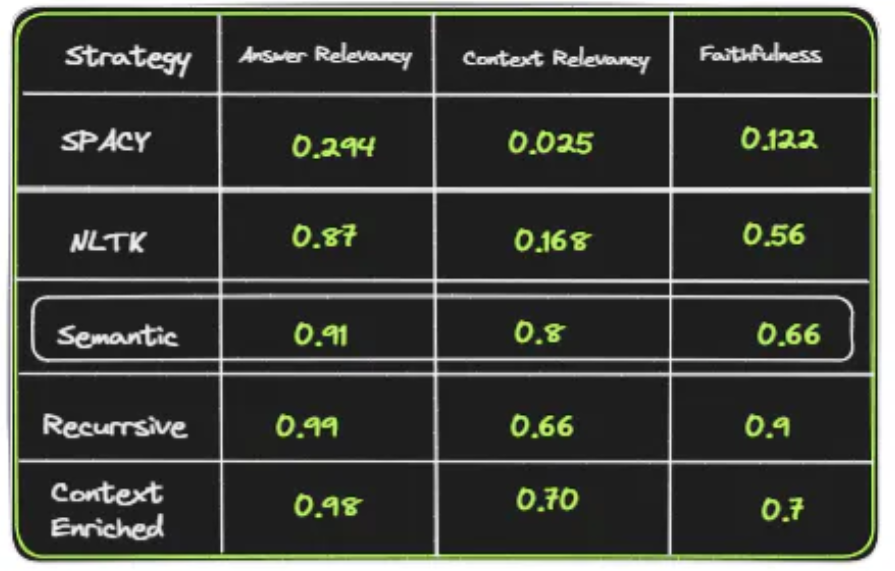
* **spaCy:** NLP görevleri için başka bir güçlü Python kütüphanesidir. Metni etkili bir şekilde ayrı cümlelere bölebilen ve sonuçta ortaya çıkan parçalarda bağlamın daha iyi korunmasını sağlayan gelişmiş bir cümle bölümlendirme özelliği sunar.



* <https://antematter.io/blogs/optimizing-rag-advanced-chunking-techniques-study>

**Chunking Stratejileri:**

* + - **NLTK:** "NLTKTextSplitter", metni alt dizelerin "listelerine" ayırmak için NLTK Tokenizers'ı kullanır ve tutarlı ve anlayışlı parçalar oluşturma sürecini kolaylaştırır. Cümle belirteci, cümle sınırlarını belirlemek için noktalama işaretlerine ve diğer dile özgü kalıplara dayalı önceden belirlenmiş kuralları kullanır.
    - **SPACY:** "SpacyTextSplitter", metni bölümlere ayırmak ve keşfedilen bölüm sınırlarıyla "Doc" nesneleri oluşturmak için Spacy'nin belirteçleyicisini kullanır ve sonuçta ortaya çıkan parçalarda daha iyi bağlam koruması sağlar. Cümle belirteci, çeşitli yazma stillerine ve geleneklerine uyum sağlarken farklı metin girdilerini ve dilleri işlemek için önceden eğitilmiş bir istatistiksel model kullanır.
    - **Semantic:** "SemanticChunker", gömme benzerliğine dayalı olarak paragraflar veya cümleler arasındaki kesme noktasını dinamik olarak seçer. Bu, her "chunk"'ın anlamsal olarak uyumlu cümleler içermesini sağlar.
    - **Recursive:** "RecursiveCharacterTextSplitter" önceden tanımlanmış ayırıcılar kullanarak metni yinelemeli olarak daha küçük parçalara ayırmayı içerir. Bu işlem, tam boyutlar değişebilse de segment boyutunda tekdüzelik elde etmeyi amaçlar.
    - **Context-Enriched (Bağlam açısından zenginleştirilmiş):** "Context-enriched" yığınlama, bilginin anlamlı bölümlere ayrılmasını ve yararlı özetler eklenmesini içerir. Düzgün bir şekilde organize edilmiş bölümler ve ayrıntılı açıklamalarla model, bilgi ortamında gezinir. Çeşitli ifadeleri keşfetmek için sorgu dönüşümünü kullanır ve her sorguda gömülü olan bağlamsal anlayışla uyumlu belgeleri alır. Bu sadece anahtar kelimelerle ilgili değil, aynı zamanda temel içerikle de ilgilidir.



Tablo, anlaşılması kolay sorulardan oluşan kolay kategorisindeki çeşitli öbekleme stratejileri için ortalama değerlendirme metriği puanlarını göstermektedir.

Semantic, veri setimizdeki kolay giriş sorguları için en iyi context-aware chunking gerçekleştirmektedir. Cevap uygunluk puanı diğerleri arasında en yüksek olmasına rağmen, bağlam uygunluk puanları diğer stratejilerin basit sorgular için bağlamla uyum sağlamada başarısız olduğunu ortaya koymaktadır.