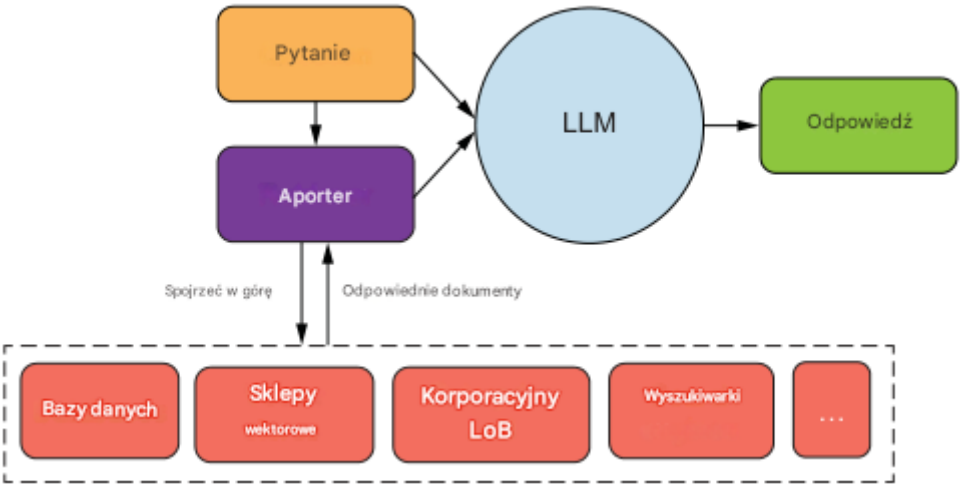
Generowanie rozszerzone o wyszukiwanie: tajna broń

Jak widzieliśmy, duże modele językowe (LLM) są bardzo potężne i pomagają nam osiągnąć rzeczy, które do niedawna były niemożliwe. Co ciekawe, LLM gromadzą wiedzę z całego świata i są dostępne dla każdego na końcu interfejsu API, w dowolnym miejscu na świecie. Jednak LLM mają ograniczenie wiedzy: ich rozumienie i wiedza rozciągają się do ostatniego punktu odcięcia szkolenia; po tej dacie nie posiadają żadnych nowych informacji. W związku z tym LLM nie mogą wykorzystywać najnowszych informacji. Ponadto korpus szkoleniowy LLM nie zawiera żadnej prywatnej, niepublicznej wiedzy. W związku z tym LLM nie mogą działać i odpowiadać na konkretne i zastrzeżone pytania przedsiębiorstw. Jednym z praktycznych sposobów rozwiązania tego problemu jest zastosowanie wzorca zwanego generowaniem rozszerzonym o wyszukiwanie (RAG).Omówimy wykorzystanie RAG do wzbogacania LLM o dane. Dowiesz się, czym jest RAG, dlaczego jest przydatny w aplikacjach korporacyjnych i jak go wdrożyć za pomocą wektorowych baz danych i indeksów. Na koniec omówione zostaną strategie fragmentacji, które optymalizują trafność i wydajność RAG. W tym rozdziale zaczniemy od zrozumienia RAG.

Czym jest RAG?

RAG to metoda łącząca dodatkowe dane z danymi wejściowymi modelu języka w celu poprawy wyników bez zmiany początkowego komunikatu. Te dane uzupełniające mogą pochodzić z bazy danych organizacji lub z zewnętrznego, zaktualizowanego źródła. Model języka przetwarza następnie połączone informacje, aby uwzględnić w swojej odpowiedzi dane faktyczne z bazy wiedzy. Ta technika jest szczególnie przydatna, gdy wymagane są najnowsze dane i ich integracja z informacjami. Z technicznego punktu widzenia, RAG łączy wstępnie wytrenowany model języka z zewnętrznym indeksem wiedzy, aby usprawnić generowanie języka. Firma Facebook AI Research po raz pierwszy przedstawiła RAG w badaniu zatytułowanym „Generowanie rozszerzone o wyszukiwanie dla zadań NLP intensywnie wykorzystujących wiedzę”. Wykazano w nim, że modele RAG mogą osiągać najnowocześniejsze wyniki w różnych zadaniach wymagających dużej wiedzy w przetwarzaniu języka naturalnego (NLP), takich jak odpowiadanie na pytania w domenie otwartej, weryfikacja faktów i wnioskowanie w języku naturalnym. Udowodniono również, że modele RAG mogą generować bardziej precyzyjny, zróżnicowany i oparty na faktach język niż wiodący model języka, który nie wykorzystuje dodatkowych danych. Model RAG łączy w sobie możliwości gęstego wyszukiwania fragmentów i modelu sekwencja-sekwencja, aby generować odpowiedzi informacyjne w oparciu o obszerny korpus tekstu. Został on zaprojektowany w celu usprawnienia systemów odpowiadania na pytania, weryfikacji faktów i generowania pytań poprzez integrację wyszukiwania informacji z generatywnymi modelami języka. Rysunek przedstawia przegląd wzorca RAG i ogólnego podejścia.



Ogólnie rzecz biorąc, istnieją dwa komponenty: wyszukiwanie informacji i generator. Jak sama nazwa wskazuje, wyszukiwanie informacji odpowiada za wyszukiwanie informacji, a generatorem jest LLM, używany do generowania tekstu. Modele podstawowe, zwłaszcza LLM, takie jak seria OpenAI GPT, posiadają ogromny potencjał, ale mają również wady. Modele te, choć potężne, cierpią na statyczną bazę wiedzy, co oznacza, że nie są świadome zdarzeń ani zmian po szkoleniu, przez co z czasem stają się nieaktualne. Są one również silnie uzależnione od danych szkoleniowych, a wszelkie stronniczości, dezinformacje lub brak równowagi w tych danych mogą wpłynąć na wynik modelu. Co więcej, LLM-om brakuje rzeczywistego zrozumienia treści, często generując tekst wyłącznie na podstawie wzorców zaobserwowanych podczas szkolenia, bez zrozumienia. Może to być problematyczne w scenariuszach korporacyjnych, gdzie obowiązują określone zasady i reguły. Wreszcie, modele te mogą generować wiarygodne, ale nieprawdziwe informacje, które mogą rozpowszechniać dezinformację bez wiarygodnej metody weryfikacji. RAG pomaga poprawić jakość odpowiedzi, czerpiąc z tych zewnętrznych źródeł wiedzy w celu uzupełnienia wewnętrznych informacji LLM-u. Jest to szczególnie pomocne w przypadku statycznej wiedzy LLM-ów, którzy nie są w stanie dostarczyć dokładnych generacji dla zdarzeń lub faktów, które miały miejsce po granicznych datach szkolenia. RAG jest niezbędnym elementem pracy z LLM-ami, wraz z szybką inżynierią. Dzięki dostępowi do szerszego zakresu informacji, RAG może generować bardziej dokładne i pouczające odpowiedzi. Gwarantuje to, że model opiera się na najbardziej aktualnych i wiarygodnych faktach, a użytkownicy mają dostęp do jego źródeł, co gwarantuje, że jego stwierdzenia można zweryfikować pod kątem poprawności i ostatecznie uznać za wiarygodne.

Korzyści z RAG

Chociaż RAG jest wciąż na wczesnym etapie rozwoju, ma potencjał, aby przekształcić krajobraz modeli generowania tekstu. RAG można wykorzystać do tworzenia bardziej kompleksowych, zróżnicowanych i opartych na faktach modeli generowania tekstu dla wielu zastosowań. W tej sekcji omówiono niezliczone korzyści, jakie mogą uzyskać przedsiębiorstwa. Możliwość pobierania danych z zewnętrznych źródeł w czasie rzeczywistym przez RAG to przełomowe rozwiązanie dla sektorów wymagających aktualnych danych, takich jak finanse, opieka zdrowotna czy wiadomości. Niezależnie od tego, czy śledzimy dynamikę rynku, aktualizujemy dokumentację medyczną, czy też śledzimy najnowsze wiadomości, RAG gwarantuje uwzględnienie najnowszych informacji. Dzięki temu wyniki są stale istotne i aktualne. W porównaniu z tradycyjnymi technikami uczenia maszynowego, RAG oferuje firmom ekonomiczną alternatywę. Tradycyjne techniki mogą wymagać ponownego trenowania modelu za każdym razem, gdy dodawane są nowe dane. Jednak dzięki RAG firmy muszą jedynie aktualizować zewnętrzny zestaw danych, oszczędzając czas i koszty związane z trenowaniem modelu i przetwarzaniem danych. RAG okazuje się szczególnie przydatny, gdy odpowiedzi wymagają cytowania danych lub wyświetlania źródeł. Może zakotwiczyć wygenerowane dane w materiale źródłowym, a nawet zapewnić cytowania. Jest to niezwykle cenne w scenariuszach akademickich, prawniczych lub zawodowych, gdzie wymagane jest precyzyjne pozyskiwanie informacji. Wszechstronność RAG obejmuje różne typy przetwarzanych danych, obsługując dane ustrukturyzowane i nieustrukturyzowane w różnych formatach. Ta adaptacyjność pozwala na wykorzystanie RAG w różnorodnych aplikacjach, od analizy złożonych zbiorów danych po przetwarzanie i generowanie treści multimedialnych. Wdrożenie RAG usprawnia interakcje z klientami i ułatwia podejmowanie lepszych decyzji. W aplikacjach obsługi klienta lub chatbotów, RAG może pobierać szczegółowe informacje z baz danych lub sekcji FAQ, co przekłada się na dokładniejsze i bardziej konstruktywne odpowiedzi. Ponadto RAG może łączyć wnioski z dużych zbiorów danych z generowaniem modeli językowych w systemach wspomagania decyzji, aby oferować kompleksowe i trafne rekomendacje. Skalowalność i wydajność RAG są wyjątkowe, umożliwiając firmom korzystanie z ogromnych zewnętrznych zbiorów danych bez przeciążania modelu językowego. Pozwala to na generowanie wyników w oparciu o szeroki zakres

informacji bez obniżania wydajności i efektywności modelu. RAG umożliwia również dostosowywanie zewnętrznych zestawów danych w oparciu o domenę biznesową. Na przykład, firma farmaceutyczna może utrzymywać zbiór danych wyłącznie na potrzeby badań nad nowymi lekami, umożliwiając RAG oferowanie odpowiedzi specyficznych dla danej dziedziny. Z perspektywy badań i rozwoju, sektory takie jak biotechnologia czy technologia mogą znacząco skorzystać z możliwości RAG w zakresie wyszukiwania odpowiednich publikacji lub danych, przyspieszając proces innowacji. RAG oferuje dynamiczne, wydajne i wszechstronne rozwiązanie do integracji zewnętrznych zestawów danych z modelami językowymi. Funkcja ta zapewnia dokładniejsze, bardziej trafne i aktualne informacje w zautomatyzowanych systemach, zwiększając wydajność, zadowolenie klientów i ułatwiając podejmowanie decyzji.

Czym jest uziemienie danych?

Uziemienie danych oznacza połączenie LLM z zewnętrznymi źródłami informacji. Uziemienie można przeprowadzić różnymi metodami, jednak powszechnie stosowaną jest metoda RAG. Zazwyczaj te zewnętrzne źródła danych są wybierane w oparciu o potrzeby danego przypadku użycia, co poprawia jakość i niezawodność generowanych danych wyjściowych. Uziemienie może poprawić generowane dane wyjściowe, dostarczając LLM informacje specyficzne dla danego przypadku użycia, istotne i nieuwzględnione w danych treningowych LLM. W ten sposób LLM może wykorzystać dane ze źródeł zewnętrznych jako kontekst i generować bardziej precyzyjne i trafne odpowiedzi dla użytkownika. Niektóre z korzyści wynikających z uziemienia to:

* Może ono pomóc LLM w tworzeniu bardziej rzeczowych i wiarygodnych danych wyjściowych, ponieważ zmniejsza ryzyko halucynacji, czyli tworzenia przez LLM fałszywych lub wprowadzających w błąd informacji w swoich danych wyjściowych.

* Może ono pomóc LLM w tworzeniu bardziej zróżnicowanych i reprezentatywnych danych wyjściowych, umożliwiając im dostęp do informacji z różnych źródeł i perspektyw oraz unikanie stronniczości lub błędów w ich wewnętrznej wiedzy.

* Może pomóc LLM-om w tworzeniu bardziej dostosowanych i spersonalizowanych wyników, umożliwiając im dostosowanie się do preferencji, potrzeb i celów użytkownika oraz dostarczanie dostosowanych rozwiązań lub sugestii.

Czym jest uziemienie danych?

Uziemienie danych oznacza połączenie LLM z zewnętrznymi źródłami informacji. Uziemienie można przeprowadzić różnymi metodami, jednak powszechnie stosowaną jest metoda RAG. Zazwyczaj te zewnętrzne źródła danych są wybierane w oparciu o potrzeby danego przypadku użycia, co poprawia jakość i niezawodność generowanych danych wyjściowych. Uziemienie może poprawić generowane dane wyjściowe, dostarczając LLM informacje specyficzne dla danego przypadku użycia, istotne i nieuwzględnione w danych treningowych LLM. W ten sposób LLM może wykorzystać dane ze źródeł zewnętrznych jako kontekst i generować bardziej precyzyjne i trafne odpowiedzi dla użytkownika. Niektóre z korzyści wynikających z uziemienia to:

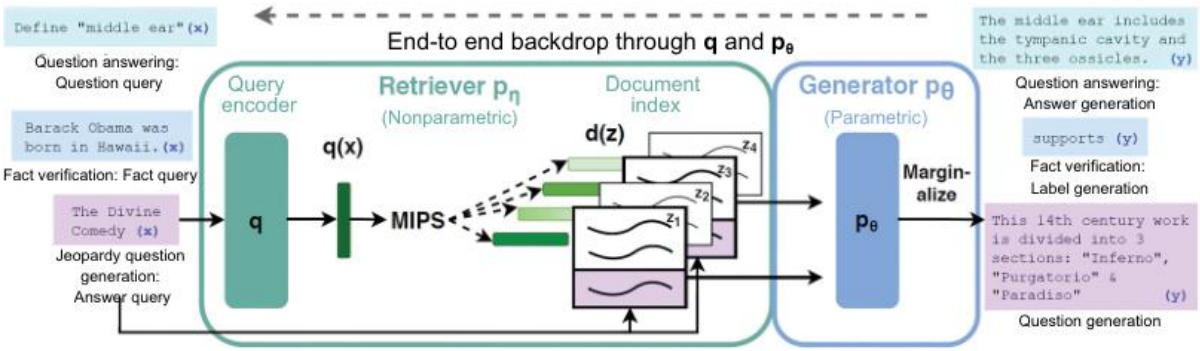
* Może ono pomóc LLM w tworzeniu bardziej rzeczowych i wiarygodnych danych wyjściowych, ponieważ zmniejsza ryzyko halucynacji, czyli tworzenia przez LLM fałszywych lub wprowadzających w błąd informacji w swoich danych wyjściowych.

* Może ono pomóc LLM w tworzeniu bardziej zróżnicowanych i reprezentatywnych danych wyjściowych, umożliwiając im dostęp do informacji z różnych źródeł i perspektyw oraz unikanie stronniczości lub błędów w ich wewnętrznej wiedzy.

* Może pomóc programom LLM w tworzeniu bardziej spersonalizowanych i dostosowanych wyników, umożliwiając im dostosowanie się do preferencji, potrzeb i celów użytkownika oraz dostarczanie rozwiązań lub sugestii dostosowanych do indywidualnych potrzeb. Modele RAG mogą wykorzystywać ogromną ilość informacji przechowywanych w korpusach tekstowych, aby wzbogacić swoje wyniki o istotne fakty i szczegóły. Mogą również obsługiwać pytania i zadania z zakresu otwartej dziedziny, wymagające rozumowania i wnioskowania wykraczającego poza zakres programów LLM. Przyjrzyjmy się architekturze RAG bardziej szczegółowo.

Architektura RAG

Wcześniej wspomniano, że architektura RAG składa się z dwóch głównych komponentów: modułu wyszukiwania (Retriever) i modelu LLM (Learning Learning Management). Moduł wyszukiwania (Retriever) wyodrębnia dane z różnych systemów przedsiębiorstwa, jak pokazano na rysunku 2 .



Komponenty te można dostosowywać i modyfikować w zależności od aplikacji i zadania, a razem zapewniają one modelowi RAG dużą elastyczność i wytrzymałość. Moduł wyszukiwania (Retriever) może uzyskiwać dostęp do informacji z prywatnych źródeł wiedzy i wyszukiwarek. Jest to mechanizm stojący za usługą Bing Chat, która pomaga dostarczać bardziej aktualne informacje. Moduł wyszukiwania (Retriever) robi więcej niż tylko wyszukiwanie — filtruje tylko istotne informacje, które stają się kontekstem dla modelu generatywnego. Wzorzec RAG łączy wyszukiwanie informacji i generowanie tekstu w celu ulepszenia wyników modelu językowego. Koder zapytania początkowo koduje pytanie lub stwierdzenie wejściowe w wektorze. Ten wektor, q(x), jest następnie wykorzystywany przez nieparametryczny moduł wyszukiwania (researcher) do przeszukiwania wstępnie skompilowanego indeksu dokumentów w celu znalezienia dokumentów odpowiadających zapytaniu. Moduł wyszukiwania wykorzystuje metodę maksymalnego iloczynu wewnętrznego (MIPS), która identyfikuje dokumenty o największym podobieństwie do wektora zapytania. Dokumenty te są wstępnie kodowane w wektorach, reprezentowanych jako d(z) w indeksie dokumentu. Generator (tj. LLM) wykorzystuje informacje z pobranych dokumentów do generowania tekstu zbliżonego do ludzkiego. Ten komponent architektury odpowiada za udzielanie odpowiedzi na pytania, weryfikację faktów lub generowanie nowych pytań. Końcowym procesem jest marginalizacja, w której model RAG, zamiast polegać na pojedynczym dokumencie w celu wygenerowania odpowiedzi, uwzględnia wszystkie istotne dokumenty. Oblicza on ogólne prawdopodobieństwo każdej możliwej odpowiedzi, sumując prawdopodobieństwa na podstawie każdego wyszukanego dokumentu, co zapewnia pełniejszą i bardziej kontekstową świadomość poprzez integrację szerokiej gamy wyszukanych informacji w końcowym tekście. Drugim kluczowym komponentem jest LLM, który pobiera kontekst z modelu wyszukiwania i generuje dane wyjściowe w języku naturalnym. Model generatywny dostarcza również informacji zwrotnej do modelu wyszukiwania, aby z czasem poprawić jego dokładność. Odbywa się to za pomocą inżynierii natychmiastowej.

System wyszukiwania

System wyszukiwania to zasadniczo komponent przeszukujący różne źródła wiedzy, jak pokazano na rysunku 2. Jego głównym celem jest przeszukiwanie korpusu informacji i znajdowanie odpowiednich informacji, które można wykorzystać. Pobrane informacje są następnie przekazywane do modelu generatora, który wykorzystuje je do generowania wyników. W architekturze RAG stosowane są dwa główne typy systemów wyszukiwania: rzadkie i gęste. Systemy wyszukiwania rzadkiego to tradycyjne systemy wyszukiwania, które wykorzystują tradycyjne techniki wyszukiwania, takie jak TF-IDF (odwrotna częstotliwość terminów i częstotliwość dokumentów), do dopasowywania zapytań do dokumentów. Systemy wyszukiwania gęstego to nowsze systemy wyszukiwania, które wykorzystują uczenie maszynowe do kodowania zapytań i dokumentów w wektorach, a następnie dopasowywania zapytań do dokumentów na podstawie podobieństwa ich wektorów. Wybór odpowiedniego typu systemu wyszukiwania w architekturze RAG (rzadkiego lub gęstego) ma kluczowe znaczenie, ponieważ zasadniczo wpływa na wydajność i przydatność modelu. Rzadkie wyszukiwarki, takie jak te wykorzystujące TF-IDF, są szybkie i wydajne, wykorzystując odwrócone indeksy do dopasowywania zapytań do dokumentów w oparciu o nakładanie się słów kluczowych. Dzięki temu nadają się do zadań wyszukiwania na dużą skalę, zależnych od słów kluczowych, przy ograniczonych zasobach obliczeniowych. Mogą jednak mieć trudności z subtelnościami języka, takimi jak synonimy i niuanse frazowania. Z kolei gęste wyszukiwarki wykorzystują techniki uczenia maszynowego do kodowania zapytań i dokumentów w wektorach, rejestrując głębsze relacje semantyczne wykraczające poza samo dopasowywanie słów kluczowych. Pozwala im to obsługiwać bardziej złożone zapytania i lepiej rozumieć kontekst, co jest szczególnie korzystne w przypadku zapytań z niejednoznacznym lub specjalistycznym językiem. Chociaż gęste wyszukiwarki często dostarczają bardziej trafne i kontekstowo poprawne dokumenty, są one bardziej intensywne obliczeniowo i wymagają znacznych ilości danych treningowych, co powoduje, że wymagają dużych zasobów zarówno w fazie treningu, jak i podczas wnioskowania. Wybór między rzadkimi a gęstymi wyszukiwarkami powinien być podyktowany specyficznymi potrzebami zadania, biorąc pod uwagę charakter zapytań, specyfikę domeny, dostępność zasobów i konieczność zrozumienia niuansów językowych. Wybór mechanizmu wyszukiwania wpływa na równowagę między wydajnością obliczeniową a głębokością zrozumienia. Pomimo kosztów obliczeniowych, gęste mechanizmy wyszukiwania są często preferowane w przypadku zadań wymagających szczegółowego zrozumienia języka. Rzadkie mechanizmy wyszukiwania mogą jednak nadal być przydatne w aplikacjach, w których szybkość i wydajność są priorytetem lub w których oczekuje się ścisłego dopasowania zapytań do tekstu dokumentu. Wybór najlepszego mechanizmu wyszukiwania dla danej aplikacji będzie zależał od jej specyficznych wymagań oraz zasobów dostępnych na wdrożenie i utrzymanie systemu.

Czym są BM25, TF-IDF i DPR?

BM25 to funkcja rankingowa używana przez wyszukiwarki do szacowania trafności dokumentów w odniesieniu do danego zapytania. Jest to jedna z najczęściej używanych funkcji rankingowych w wyszukiwaniu informacji. BM25 uwzględnia wiele czynników, w tym częstotliwość występowania (TF) terminów w zapytaniu w dokumencie, odwrotną częstotliwość występowania (IDF) tych terminów w zapytaniu oraz długość dokumentu.

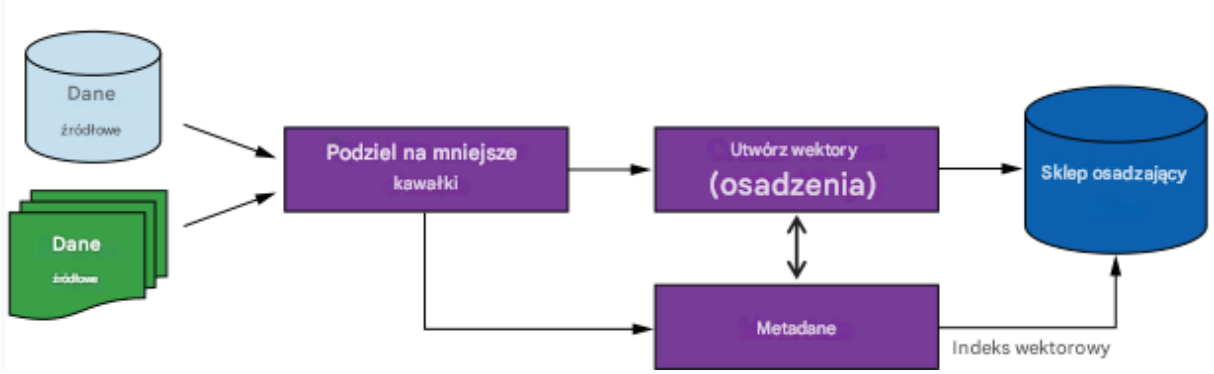
TF-IDF to miara statystyczna używana do oceny ważności słowa dla dokumentu w zbiorze dokumentów. Wartość TF-IDF rośnie proporcjonalnie do liczby wystąpień słowa w dokumencie. Maleje proporcjonalnie do liczby dokumentów w zbiorze, które zawierają to słowo. TF-IDF jest często używana w wyszukiwaniu informacji i eksploracji tekstu do klasyfikowania dokumentów na podstawie ich trafności dla danego zapytania.

DPR to model sieci neuronowej, który pobiera istotne fragmenty z dużego korpusu tekstowego. Jest on trenowany na ogromnym zbiorze danych tekstowych i kodowych i uczy się osadzać fragmenty i zapytania w gęstej przestrzeni wektorowej. DPR może wyszukiwać fragmenty semantycznie, podobnie jak zapytanie, poprzez obliczenie podobieństwa cosinusowego między wektorami fragmentu i zapytania.

BM25 i TF-IDF to statystyczne miary istotności dokumentu dla danego zapytania. BM25 uwzględnia jednak dodatkowe czynniki, takie jak długość dokumentu i nasycenie częstotliwością występowania terminów. DPR można wykorzystać do poprawy wydajności funkcji rankingowych BM25 i TF-IDF.

Ogólnie rzecz biorąc, musimy postępować zgodnie z procesem przedstawionym na rysunku 3, aby wykorzystać potencjał modeli LLM w naszych danych. Źródło pobierane przez program do wyszukiwania informacji musiałoby zostać podzielone na mniejsze fragmenty. Jest to konieczne, aby informacje były łatwiejsze w zarządzaniu i zgodne z oknami kontekstowymi modeli LLM. Następnie musimy utworzyć osadzenia tych mniejszych fragmentów i połączyć je ze źródłem jako metadane. Na koniec te osadzenia i powiązane metadane powinny zostać zapisane w magazynie danych.

Ogólnie rzecz biorąc, musimy postępować zgodnie z procesem przedstawionym na rysunku 3, aby wykorzystać potencjał modeli LLM w naszych danych.



Źródło pobierane przez program do wyszukiwania informacji musiałoby zostać podzielone na mniejsze fragmenty. Jest to konieczne, aby informacje były łatwiejsze w zarządzaniu i zgodne z oknami kontekstowymi modeli LLM. Następnie musimy utworzyć osadzenia tych mniejszych fragmentów i połączyć je ze źródłem jako metadane. Na koniec te osadzenia i powiązane metadane powinny zostać zapisane w magazynie danych. Aby RAG był wydajny i skalowalny, komponent wyszukiwania musi szybko pobierać najistotniejsze dokumenty spośród potencjalnie miliardów kandydatów. Aby sprostać temu wyzwaniu, potrzebujemy dwóch komponentów: wektorowej bazy danych i indeksu. Wektorowa baza danych to system, który przechowuje i zapewnia dostęp do danych ustrukturyzowanych lub nieustrukturyzowanych (np. tekstu lub obrazów) wraz z ich osadzeniem wektorowym, które stanowi numeryczną reprezentację danych. Indeks wektorowy to struktura danych, która umożliwia wydajne i szybkie wyszukiwanie najbliższych sąsiadów w wielowymiarowej przestrzeni wektorowej. Bez wydajnych wektorowych baz danych i indeksów, etap wyszukiwania stałby się wąskim gardłem, spowalniając i utrudniając działanie całego systemu RAG. Dzięki tym narzędziom możliwe jest wyszukiwanie istotnych dokumentów w czasie rzeczywistym, co pozwala komponentowi generowania szybko generować odpowiedzi i czyni system użytecznym w aplikacjach takich jak udzielanie odpowiedzi na pytania w domenie otwartej.

Zrozumienie wektorowych baz danych

Wektorowe bazy danych umożliwiają przedsiębiorstwom zarządzanie, zabezpieczanie i skalowanie osadzenia w środowisku produkcyjnym. W wielu przedsiębiorstwach wektorowe bazy danych do zastosowań wyszukiwania semantycznego rozwiązują wymagania dotyczące wydajności i bezpieczeństwa wymagane w systemach produkcyjnych. Wektorowa baza danych jest specjalnie zaprojektowana do działania na wektorach osadzania. Wraz ze wzrostem popularności LLM i generatywnej sztucznej inteligencji (AI) w ostatnim czasie wzrosło również wykorzystanie osadzenia do kodowania danych nieustrukturyzowanych. Wektorowe bazy danych stały się skutecznym rozwiązaniem dla przedsiębiorstw, umożliwiającym dostarczanie i skalowanie takich przypadków użycia. Wektorowe bazy danych to specjalistyczne bazy danych, które przechowują dane jako wielowymiarowe wektory i ich oryginalną zawartość. Oferują one możliwości zarówno indeksów wektorowych, jak i tradycyjnych baz danych, takie jak zoptymalizowane przechowywanie, skalowalność, elastyczność i obsługa języków zapytań. Umożliwiają użytkownikom wyszukiwanie i pobieranie podobnych lub istotnych danych na podstawie ich znaczenia semantycznego lub kontekstowego. Biorąc pod uwagę ogromną liczbę dokumentów w dużych korpusach, siłowe porównanie wektora zapytania z każdym wektorem dokumentu jest obliczeniowo nieopłacalne. Rozwiązaniem jest wyszukiwanie wektorowe, obejmujące indeksy i bazy danych, które umożliwiają efektywne przechowywanie i wyszukiwanie w pobliżu sąsiadów w przestrzeniach wielowymiarowych. Rysunek 4 przedstawia typowy schemat implementacji bazy danych wektorowych podczas implementacji wzorca RAG z modelami LLM.

Bazy danych wektorowych mogą pomóc modelom RAG szybko znaleźć dokumenty lub fragmenty najbardziej podobne do danego zapytania i wykorzystać je jako dodatkowy kontekst dla modelu LLM. W zależności od kompromisu między szybkością a dokładnością, bazy danych wektorowych mogą również obsługiwać różne strategie wyszukiwania, takie jak metody dokładne, przybliżone lub hybrydowe. Posiadanie bazy danych wektorowych to dobry punkt wyjścia, ale znalezienie najbardziej podobnych dokumentów lub fragmentów jest możliwe tylko za pomocą indeksu wektorowego.

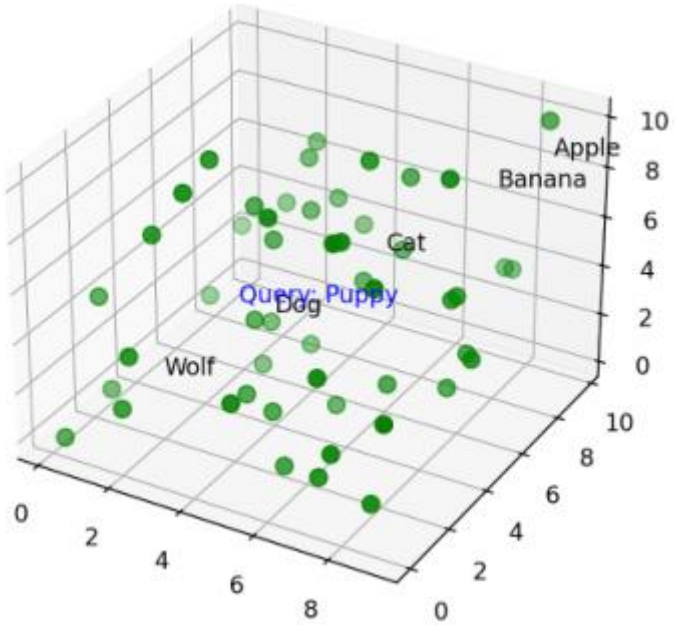
Czym jest indeks wektorowy?

Indeks wektorowy to struktura danych w bazie danych wektorowych zaprojektowana w celu zwiększenia wydajności przetwarzania, szczególnie przydatna w przypadku wielowymiarowych danych wektorowych spotykanych w modelach LLM. Jego funkcją jest usprawnienie procesów wyszukiwania i wyszukiwania w bazie danych. Dzięki implementacji indeksu wektorowego system jest w stanie przeprowadzać szybkie wyszukiwanie podobieństw, identyfikując wektory, które ściśle pasują lub są najbardziej podobne do danego wektora wejściowego. Zasadniczo indeksy wektorowe mają na celu umożliwienie szybkiego i precyzyjnego wyszukiwania podobieństw, ułatwiając odzyskiwanie osadzeń wektorowych. Organizują wektory za pomocą różnych technik, takich jak haszowanie, klasteryzacja czy metody oparte na drzewach, aby ułatwić znalezienie najbardziej podobnych wektorów na podstawie ich metryk odległości lub podobieństwa. Na przykład FAISS (Facebook AI Similarity Search) to popularny indeks wektorów, który efektywnie obsługuje miliardy wektorów. Aby utworzyć indeksy wektorów dla swoich osadzeń, istnieje wiele opcji, takich jak dokładne lub przybliżone algorytmy najbliższego sąsiedztwa (np. HNSW lub IVF), różne metryki odległości (np. cosinus lub euklidesowa) lub różne techniki kompresji (np. kwantyzacja lub przycinanie). Twoja metoda indeksowania zależy od zrównoważenia szybkości, dokładności i zużycia pamięci. Możemy użyć różnych metod

matematycznych, aby porównać podobieństwo dwóch osadzeń wektorów — są one przydatne podczas wyszukiwania i dopasowywania różnych osadzeń. Zobaczmy, co oznacza wyszukiwanie wektorów i jak możemy zastosować różne funkcje matematyczne podczas wyszukiwania.

Wyszukiwanie wektorowe

Wyszukiwanie wektorowe to operacja zapytania, która znajduje wektory najbardziej podobne do danego wektora zapytania na podstawie metryki podobieństwa. We wzorcu RAG dla LLM indeks wektora przechowuje osadzenia lub fragmenty dokumentów, które LLM może pobrać jako kontekst do generowania odpowiedzi. Wyszukiwanie wektorowe służy do znajdowania dokumentów lub fragmentów najbardziej odpowiednich dla zapytania na podstawie podobieństwa między wektorem zapytania a wektorami dokumentów w indeksie. Miary podobieństwa to metody matematyczne, które porównują dwa wektory i obliczają wartość odległości między nimi. Wartość ta wskazuje, jak bardzo oba wektory różnią się lub są podobne pod względem znaczenia semantycznego. Odległość może być oparta na wielu kryteriach, takich jak długość odcinka między dwoma punktami, kąt między dwoma kierunkami lub liczba niedopasowanych elementów w dwóch tablicach. Miary podobieństwa są przydatne w zadaniach uczenia maszynowego obejmujących grupowanie lub klasyfikowanie obiektów danych, szczególnie w przypadku wyszukiwania wektorowego lub semantycznego. Na przykład, jeśli chcemy znaleźć słowa podobne do słowa „szczeniak", możemy wygenerować osadzenie wektorowe dla tego słowa i poszukać innych słów z bliskimi osadzeniami wektorowymi, takich jak „pies" (rysunek 5).



Powinniśmy wybrać miarę podobieństwa, która najlepiej odpowiada potrzebom danych i zapytań danego przypadku użycia. Musimy użyć miary podobieństwa, aby przeprowadzić wyszukiwanie wektorowe, matematyczną metodę obliczania odległości między dwoma wektorami. Im mniejsza odległość, tym bardziej podobne są wektory. Niektóre popularne usługi gotowe do użycia w przedsiębiorstwach, takie jak Azure AI Search, obsługują kilka miar podobieństwa. Niektóre z bardziej popularnych wyszukiwań podobieństwa to:

* Podobieństwo cosinusowe — ta miara oblicza cosinus kąta między dwoma wektorami. Jego zakres wynosi od -1 do 1, gdzie 1 oznacza wektory identyczne, a -1 wektory przeciwne. Podobieństwo cosinusowe jest powszechnie stosowane w przypadku znormalizowanych przestrzeni osadzania.

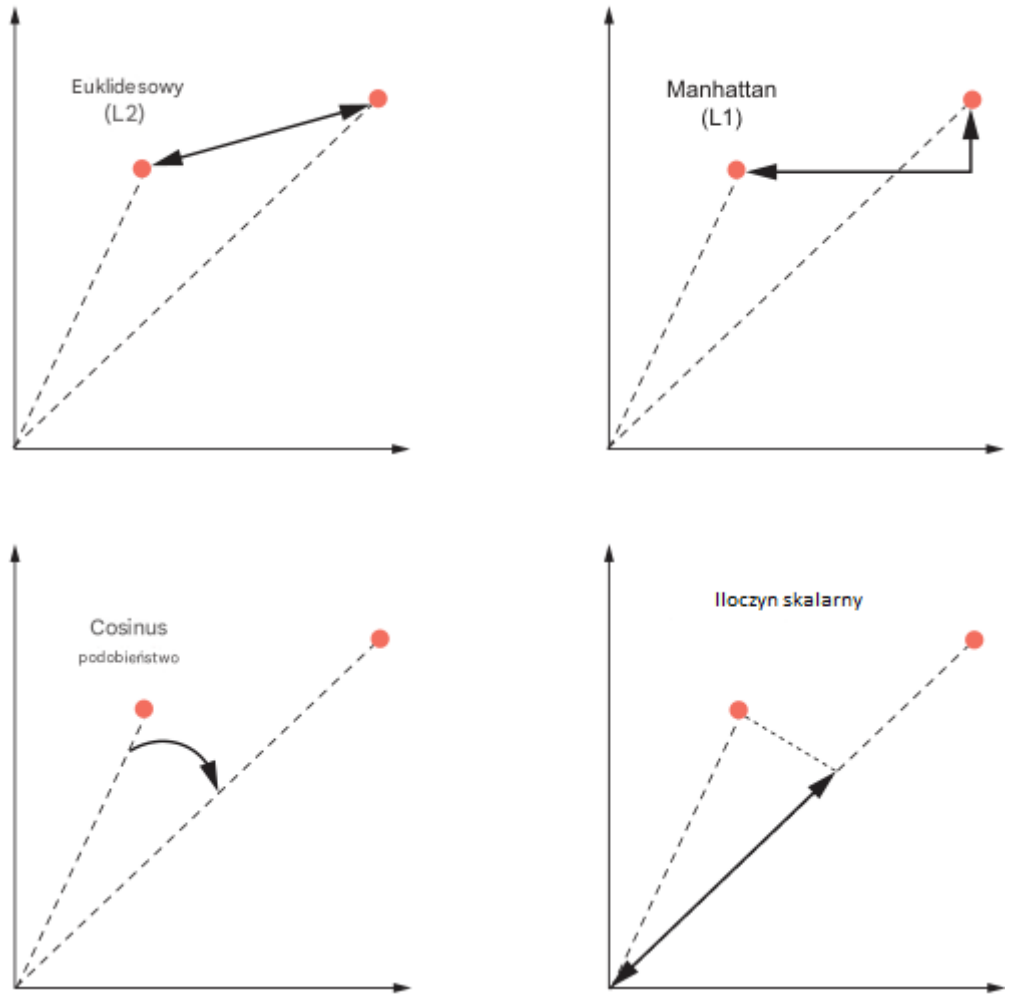
* Odległość euklidesowa kwadratowa lub odległość L2 kwadrat — oblicza odległość w linii prostej między dwoma wektorami. Jej zakres wynosi od 0 do nieskończoności [0, ∞], gdzie 0 oznacza wektory identyczne, a większe wartości oznaczają wektory bardziej odmienne. Kwadratowa odległość euklidesowa jest również znana jako norma L2.

* Iloczyn skalarny — ta miara oblicza iloczyn długości dwóch wektorów i cosinusa kąta między nimi. Jej zakres wynosi od –nieskończoności do nieskończoności [–, ∞], gdzie 0 oznacza wektory ortogonalne, a większe wartości oznaczają wektory bardziej podobne. Iloczyn skalarny jest równoważny podobieństwu cosinusowemu dla znormalizowanych przestrzeni zanurzenia, ale jest bardziej efektywny.

* Odległość Hamminga — ta miara oblicza liczbę różnic między wektorami w każdym wymiarze.

* Odległość Manhattanu lub L1 — ta miara mierzy sumę bezwzględnych różnic między współrzędnymi dwóch wektorów. Jej zakres wynosi od 0 do nieskończoności [0, ∞], gdzie 0 oznacza wektory identyczne, a większe wartości oznaczają wektory przeciwne, czyli wektory niepodobne.

Rysunek 6 przedstawia różne miary podobieństwa.



Ważne jest, aby użyć tej samej metryki, na której trenowano model bazowy. Na przykład, w przypadku klasy modeli GPT OpenAI, funkcją odległości jest podobieństwo cosinusowe.

UWAGA: Osadzenia OpenAI są znormalizowane do długości 1, co oznacza, że moduł każdego wektora jest równy 1. Dlatego, jeśli użyjemy osadzeń OpenAI znormalizowanych do długości 1, możemy wybrać podobieństwo cosinusowe lub odległość euklidesową jako funkcję odległości, a otrzymamy te same wyniki dla wyszukiwania wektorów. Jednak podobieństwo cosinusowe może być nieco szybsze do obliczenia, ponieważ wymaga jedynie operacji iloczynu skalarnego.

Wybór właściwej miary odległości zależy od konkretnego przypadku użycia, charakteru danych i oczekiwanych rezultatów. Tabela 1 przedstawia krótki przegląd tego, kiedy należy użyć każdej z miar.

Pomiar: Kiedy stosować: Zaleta: Wada

Podobieństwo cosinusowe: Idealne do porównania tekstu i dokumentów, gdzie wielkość wektorów nie jest tak ważna jak orientacja; powszechne w zadaniach przetwarzania języka naturalnego. : Skuteczne w przestrzeniach wielowymiarowych i dla wektorów znormalizowanych; ignoruje wielkość wektorów, koncentrując się na orientacji, dzięki czemu nadaje się do porównywania dokumentów o różnej długości: Nie jest skuteczne, jeśli wielkość wektorów jest istotna.

Kwadrat euklidesowy (L2): Odpowiednie dla danych geometrycznych lub przestrzennych, np. w przetwarzaniu obrazów lub podczas klasteryzacji wielowymiarowych danych liczbowych: Odzwierciedla rzeczywistą odległość między punktami w przestrzeni euklidesowej, dzięki czemu jest intuicyjne i odpowiednie dla przestrzennych zbiorów danych. : Może być wrażliwe na skalę danych. Wysokie wymiary mogą prowadzić do wady wymiarowości.

Iloczyn skalarny: Efektywne w przypadku danych o dużej objętości i wielu wymiarach, takich jak preferencje użytkowników w systemach rekomendacji: Efektywne obliczeniowo, szczególnie w przypadku wektorów rzadkich. Sprawdza się w przypadkach, gdy wielkość wektorów ma znaczenie. Interpretacja jest mniej intuicyjna niż podobieństwo cosinusowe i może być wrażliwa na wielkości wektorów.

Odległość Hamminga: Najlepsza do porównywania danych binarnych lub kategorycznych, takich jak sekwencje genetyczne lub wykrywanie błędów w transmisji danych:

Prosta i skuteczna w przypadku zbiorów danych z atrybutami dyskretnymi: Dotyczy tylko ciągów o równej długości i nie uwzględnia wielkości różnic.

Odległość Manhattan (L1): Przydatna w wyszukiwaniu ścieżek w formie siatki (np. w układach dróg miejskich) oraz w przypadkach, gdy istotne są różnice w poszczególnych wymiarach: Jest bardziej wrażliwa na różnice w poszczególnych wymiarach niż odległość L2; odporna na wartości odstające.: Może nie odzwierciedlać rzeczywistej odległości w przestrzeniach niesiatkowych lub w danych wielowymiarowych.

Wyzwania związane z RAG

Przedsiębiorstwa rozważające wdrożenie systemów RAG napotykają na szereg przeszkód, które wymagają starannego rozważenia. Przede wszystkim kluczowe jest zapewnienie efektywnej skalowalności wraz ze wzrostem wolumenu danych. Wraz ze wzrostem danych rośnie również złożoność i rozmiar indeksu wyszukiwania. Zarządzanie tym wzrostem staje się trudne i wymaga bardziej wydajnych zasobów obliczeniowych. W szczególności gęste systemy wyszukiwania, które wymagają dużych zasobów obliczeniowych i pamięci masowej, wymagają starannego zrównoważenia, aby zapewnić skalowalność. Ponadto utrzymanie wydajnego i szybkiego indeksu wyszukiwania staje się kluczowe wraz ze wzrostem wolumenu dokumentów. Równoległe przetwarzanie żądań,

zarządzanie mechanizmami ponawiania prób i wdrożenie odpowiedniej infrastruktury są niezbędne do osiągnięcia skalowalności systemów RAG.

Zapewnienie jakości i trafności indeksowanych danych to kolejny istotny problem. Użyteczność systemu RAG zależy od jakości jego danych; nieaktualne lub nieistotne informacje będą prowadzić do odpowiedzi poniżej oczekiwań – zasada „śmieci na wejściu – śmieci na wyjściu" nadal obowiązuje. Podkreśla to potrzebę starannej konserwacji i regularnej aktualizacji indeksu dokumentów, aby dostosować go do zmieniających się wymagań przedsiębiorstwa.

Po wdrożeniu systemy RAG wprowadzają dodatkową warstwę złożoności w integracji istniejących przepływów pracy, wymagając ciągłej konserwacji w celu zapewnienia spójnej wydajności. Systemy RAG muszą być płynnie zintegrowane z istniejącym środowiskiem technicznym przedsiębiorstwa. Proces ten często wiąże się z koniecznością radzenia sobie ze złożonymi problemami zarządzania danymi i zapewnienia interoperacyjności systemów. Systemy RAG wymagają złożonego kodowania i wykonywania zapytań dotyczących gęstych wektorów w czasie rzeczywistym, co może powodować opóźnienia i wpływać na czas reakcji. W przypadku aplikacji wymagających szybkich odpowiedzi, takie opóźnienie może nie spełniać oczekiwań użytkowników co do szybkości. Ponadto, złożona natura modeli RAG utrudnia identyfikację przyczyn błędów. Skuteczne wyszukiwanie i naprawianie błędów jest ważne, niezależnie od tego, czy występują one podczas pobierania, czy generowania. Co więcej, po wdrożeniu systemy RAG wprowadzają dodatkową złożoność podczas integracji z istniejącymi przepływami pracy. Zapewnienie płynnej integracji z technicznym środowiskiem przedsiębiorstwa wiąże się z koniecznością radzenia sobie z problemami zarządzania danymi i zgodnością systemów. Z perspektywy społeczno-technicznej, zapewnienie uczciwości i bezstronności systemów RAG jest koniecznością. Ryzyko utrwalenia istniejących uprzedzeń z danych treningowych jest realne i może mieć dalekosiężne konsekwencje, wymagając rygorystycznego nadzoru i strategii ograniczania ryzyka. Ponadto prywatność i bezpieczeństwo są również kluczowe, zwłaszcza jeśli indeksowane dane zawierają informacje poufne, co wymusza ścisłe przestrzeganie przepisów o ochronie danych. Fragmentacja to kluczowy problem, który musi zostać rozwiązany przez implementację RAG. Fragmentacja to podział długiego tekstu na mniejsze segmenty, które LLM może łatwiej obsłużyć. Może to pomóc zmniejszyć zapotrzebowanie modelu na moc obliczeniową i pamięć oraz poprawić jakość i trafność tekstu wyjściowego. Fragmentacja może również pomóc modelowi skoncentrować się na najważniejszych częściach tekstu i uniknąć fragmentów nieistotnych lub powtarzających się. Trudności związane z fragmentacją są ogromne; omówimy je szczegółowo w kolejnych sekcjach. Przedsiębiorstwa muszą być świadome tych wyzwań i rozważyć je w kontekście korzyści, jakie mogą przynieść systemy RAG, takie jak zwiększona dokładność i trafność kontekstowa w zadaniach przetwarzania języka naturalnego. Wdrażając rozwiązanie bazujące na RAG, należy wziąć pod uwagę kompromisy dotyczące kosztów, zasobów i potencjalnego ryzyka.

Pokonywanie wyzwań związanych z fragmentacją

Współcześnie przedsiębiorstwa napotykają wiele wyzwań podczas wdrażania RAG na skalę produkcyjną. Jak wspomniano wcześniej, fragmentacja to proces dzielenia długiej sekwencji tekstu na mniejsze, łatwiejsze w zarządzaniu fragmenty. Jest to niezbędne w przypadku modeli LLM, które mają ograniczoną moc przetwarzania. Modele RAG zazwyczaj wykorzystują algorytm fragmentacji do podziału tekstu wejściowego na mniejsze fragmenty, które następnie są przetwarzane przez LLM. LLM generuje odpowiedź dla każdego fragmentu, a następnie odpowiedzi są łączone w celu utworzenia ostatecznego wyniku. Fragmentacja może jednak stanowić wyzwanie dla modeli RAG z następujących powodów:

* Fragmenty mogą nie być wyrównane z naturalnymi granicami tekstu. Może to prowadzić do generowania przez LLM odpowiedzi niepoprawnych gramatycznie lub semantycznie.

* Fragmenty mogą różnić się długością i złożonością. Może to utrudniać LLM generowanie odpowiedzi o spójnej jakości.

* Fragmenty mogą zawierać wiele intencji. Może to utrudniać LLM identyfikację właściwej intencji i wygenerowanie odpowiedniej odpowiedzi.

Zaczynamy od zrozumienia strategii dzielenia na fragmenty.

Strategie fragmentacji

Jedną z wad wyszukiwania jest to, że w oknie kontekstowym możemy umieścić tylko ograniczoną ilość informacji. Jeśli użyjemy modeli OpenAI jako miary, w zależności od modelu, możemy użyć tylko skończonego zbioru informacji, który można przekazać, jak pokazano w tabeli 2.

Open AI model	Maximum length (token size)
GPT-3.5 Turbo	4K tokens; approx. 5 pages
GPT-4	8K tokens; approx. 10 pages
GPT-4 32K	32K tokens; approx. 40 pages
GPT-4 Turbo, GPT-4o	128K tokens; approx. 300 pages

W praktyce ta długość jest jeszcze krótsza, biorąc pod uwagę, że potrzebujemy miejsca na generowanie. To właśnie tutaj fragmentacja staje się kluczowa. Podział na fragmenty oznacza dzielenie dużych dokumentów lub fragmentów tekstu na mniejsze, łatwiejsze do przyswojenia części lub fragmenty. Ma to na celu przyspieszenie i usprawnienie procesu wyszukiwania, szczególnie w przypadku pracy z ogromnymi zbiorami tekstów; głównym powodem jest również ograniczenie okna kontekstowego w przypadku LLM. Podział na fragmenty jest przydatny w RAG z kilku powodów: * Granularność — podczas przeszukiwania dużego korpusu w celu uzyskania istotnych informacji, wyszukiwanie na poziomie granularności mniejszych fragmentów może prowadzić do bardziej precyzyjnych wyszukiwań niż przeszukiwanie całych dokumentów. Może to poprawić ogólną jakość odpowiedzi generowanych przez RAG.

* Wydajność — praca z mniejszymi fragmentami może zwiększyć wydajność procesu wyszukiwania, szczególnie w przypadku stosowania gęstych wyszukiwarek, które osadzają każdy fragment w wielowymiarowej przestrzeni wektorowej.

* Elastyczność — podział na fragmenty pozwala systemowi dopasować do danego zapytania informacje o różnej długości, oferując większą elastyczność w zakresie tego, co jest uznawane za istotne. Rozważając strategię fragmentacji, musimy spojrzeć na nią holistycznie i sprawdzić, w jaki sposób wyniki wyszukiwania oddają istotę zapytania użytkownika. Zbyt duży lub zbyt mały fragment może prowadzić do niedokładnych wyników. Zasadniczo, jeśli fragment ma sens dla nas, ludzi, bez dodatkowych informacji, LLM również mógłby go zrozumieć. W przypadku konwersacji, gdy następują zwroty akcji i dialog się wydłuża, ważne jest, aby ocenić, ile z poprzedniej rozmowy jest potrzebne do następnej tury w bieżącym kontekście. Dodawanie większych fragmentów może wpłynąć na trafność, a także uwzględnimy ograniczenia okien kontekstowych LLM. Wykorzystajmy scenariusz z chatbotem

obsługi klienta u dostawcy usług — na przykład operatora telefonii komórkowej. Rozmowa może rozpocząć się od jednego tematu, pytania o aktywację nowego telefonu, i może przejść do innych tematów, takich jak szczegóły dotyczące planów, produktów dodatkowych, szczegółów zasięgu, rozliczeń, metod płatności itp. W tym przykładzie, gdy rozmowa przechodzi z jednego tematu na drugi, w wielu przypadkach nie potrzebujemy całej poprzedniej historii i dialogu, i można je albo odrzucić, albo przyciąć. Oczywiście w niektórych przypadkach chcielibyśmy tylko niezbędnych szczegółów dla kontekstu trwającej rozmowy. Długość dzielonych informacji zależy od przypadku użycia i oczekiwanego zachowania użytkownika. Na przykład, jeśli podzielimy akapit, otrzymamy reprezentację wektorową, która przechwyci więcej znaczenia z treści. Różni się to od osadzania na poziomie zdań, gdzie reprezentacja wektorowa odzwierciedla więcej znaczenia zdania. Prowadziłoby to do porównań innych zdań i byłoby bardziej ograniczone niż poprzednie podejście oparte na akapitach.

Czynniki wpływające na strategie fragmentacji

Zanim przejdziemy do omówienia różnych podejść do fragmentacji, kilka dodatkowych czynników, które należy wziąć pod uwagę z perspektywy strategii fragmentacji, pomoże nam zrównoważyć wyższą dokładność z utrzymaniem akceptowalnych progów wydajności i kosztów:

* Charakter treści — Charakter indeksowanej treści wpływa na strategię fragmentacji. Na przykład krótsze treści, takie jak tweety, mogą wymagać innego fragmentacji niż dłuższe treści, takie jak książki czy raporty. Krótsze treści mogą być fragmentowane razem, podczas gdy dłuższe treści, takie jak dokumenty, raporty i podobne, mogą wymagać podziału na mniejsze części w celu wydajnego przetwarzania.

* LLM i powiązany model osadzania — LLM i powiązany model osadzania mogą również wpływać na strategię fragmentacji. Na przykład niektóre modele mogą być bardziej wydajne w przetwarzaniu mniejszych fragmentów lub, biorąc pod uwagę swoją architekturę, fragmentów o określonym rozmiarze, podczas gdy inne mogą lepiej obsługiwać większe fragmenty. Ważne jest, aby wiedzieć, którego LLM i powiązanego modelu osadzania użyjemy. Na przykład, korzystając z OpenAI, powinniśmy rozważyć osadzanie tekstu w formacie ada-002 o rozmiarze 256 lub 512 tokenów.

* Złożoność zapytania — długość i złożoność zapytania użytkownika mogą wpływać na podejście do fragmentacji. Bardziej złożone zapytania mogą wymagać bardziej złożonych strategii fragmentacji, aby dopasować zapytanie do odpowiednich danych. Należy pamiętać, że LLM nie są wyszukiwarkami i nie powinny być używane jako takie. Złożoność zapytania jest wielowymiarowa, zarówno pod względem długości, jak i złożoności, i może wiązać się z podziałem zapytania na mniejsze podzapytania, które dotyczą różnych aspektów pierwotnego zapytania, zanim wszystko zostanie sprowadzone do odpowiedzi. Na przykład zapytanie „Jaka jest stolica Wielkiej Brytanii?" jest bardzo konkretne i proste. Natomiast zapytanie „Jakie są ekonomiczne implikacje rozwoju sztucznej inteligencji w różnych branżach w Stanach Zjednoczonych?" jest wieloaspektowe. Wymaga ono głębszego zrozumienia technologii (w tym przykładzie sztucznej inteligencji) oraz szczegółów geograficznych i branżowych, aby wywnioskować znaczenie implikacji.

* Integracja z aplikacją — Zrozumienie, w jaki sposób dane wyjściowe (wynik zapytania) są wykorzystywane w aplikacji, może również wpłynąć na strategię fragmentacji. Na przykład ograniczenia LLM i okna kontekstowego mogą dyktować, jak dane powinny być fragmentowane, aby uzyskać najlepsze rezultaty. Uwzględnia to również inne dane i metadane, których aplikacja może potrzebować.

* Wstępne przetwarzanie danych — Wstępne przetwarzanie danych pomoże zwiększyć jakość generowania i pomoże nam określić potencjalnie dobry rozmiar. Wstępne przetwarzanie

obejmowałoby usuwanie zbędnego szumu lub wykorzystanie innych technik sztucznej inteligencji do wyodrębnienia informacji, w tym czyszczenie danych, transformację danych z jednego formatu do innego, normalizację cech (w razie potrzeby), tokenizację, usuwanie często używanych słów kluczowych (takich jak „jest”, „ten”, „i”) itd.

* Ocena i porównanie różnych rozmiarów fragmentów — Kluczowe jest, aby ocenić i porównać wpływ różnych rozmiarów fragmentów zarówno na jakość, jak i wydajność procesu. Może to być szczególnie ważne w środowiskach korporacyjnych, gdzie różne rozmiary fragmentów mogą być używane w zależności od charakteru treści i konieczne może być znalezienie równowagi między dokładnością a wydajnością. Ocena ta będzie uwzględniać zarówno jakość, jak i wydajność.

Można zastosować kilka podejść do dzielenia informacji na fragmenty, jak przedstawiono w tabeli 3. Warto zauważyć, że idealna strategia dzielenia może się różnić w zależności od korpusu, charakteru zapytań i specyficznych wymagań aplikacji. Znalezienie najskuteczniejszego podejścia dla konkretnej implementacji RAG może wymagać eksperymentów.

Podejście oparte na fragmentacji: Opis

Stała długość: Podzielenie dokumentów na fragmenty o ustalonej liczbie słów lub tokenów. Jest to proste, ale czasami może powodować rozdzielenie informacji, które w idealnym przypadku powinny być przechowywane razem.

Przesuwne okno: Zastosowanie przesuwnego okna o stałej wielkości z nakładającymi się danymi lub bez. Pozwala to uniknąć pominięcia ważnych granic w tekście, ale może również prowadzić do redundancji, jeśli występuje znaczne nakładanie się.

Oparte na interpunkcji: Podzielenie tekstu na podstawie znaków interpunkcyjnych, takich jak akapity lub sekcje. Jest to mniej arbitralne niż podział na fragmenty o stałej długości i często zachowuje integralność semantyczną treści. Może jednak prowadzić do zmiennych rozmiarów fragmentów.

Podziały tematyczne lub sekcyjne: W ustrukturyzowanych dokumentach, takich jak artykuły w Wikipedii, naturalne podziały, takie jak sekcje lub podsekcje, mogą być używane do definiowania fragmentów. Ta metoda zapewnia spójność semantyczną treści w obrębie fragmentu.

Adaptacyjne: Wykorzystanie algorytmów lub modeli, które adaptacyjnie określają najlepszy sposób fragmentacji dokumentów na podstawie ich treści. Może to być bardziej złożone, ale może dać semantycznie spójne fragmenty.

W zależności od rozmiaru i struktury tekstu, istnieją różne sposoby jego fragmentowania w metodzie RAG. Niektóre z popularnych metod to:

* Podział zdań — Jak sama nazwa wskazuje, granice zdań służą do podziału tekstu, co jest przydatne, aby zapewnić, że każdy fragment zawiera całe zdania, zachowując kontekst i znaczenie.

* Podział o stałej długości — W tym przypadku tekst jest dzielony na fragmenty o stałej długości. Może to czasami skutkować ucięciem zdań w środku.

* Podział na tokeny — Podział tekstu na podstawie stałej liczby tokenów (np. słów). Jest to bardziej precyzyjne niż podział zdań, ale nadal może skutkować ucięciem zdań.

* Semantyczny podział na fragmenty — Wykorzystanie narzędzi przetwarzania języka naturalnego (NLP) do identyfikacji spójnych segmentów w tekście. Na przykład podział tekstu na podstawie tematów lub akapitów.

* Hierarchiczny podział na fragmenty — Podział tekstu na hierarchiczne sekcje, takie jak rozdziały, sekcje i podsekcje.

Aby zilustrować, jak różne podejścia do fragmentacji (o stałej długości i semantyczne przetwarzanie języka naturalnego) mogą wpływać na wyniki, jako tekst wejściowy posługujemy się przykładem Konstytucji Wielkiej Brytanii z Wikipedii . Wynik widać na rysunku 7 po zastosowaniu podejścia do fragmentacji o stałej długości. Tekst jest dzielony na fragmenty o stałej długości, a w tym prostym przykładzie widzimy, że niektóre informacje są ucięte, a kontekstu brakuje.

The constitution of the United Kingdom or British constitution comprises the written and unwritten arrangements that establish the United Kingdom of Great Britain and Northern Ireland as a political body. Unlike in most countries, no attempt has been made to codify such arrangements into a single document, thus it is known as an uncodified constitution. This enables the constitution to be easily changed as no provisions are formally entrenched;[2] the Supreme Court of the United Kingdom recognises that there are constitutional principles, including parliamentary sovereignty, the rule of law, democracy, and upholding international law.[3]

The Supreme Court also recognises that some Acts of Parliament have special constitutional status, and are therefore part of the constitution.[4] These include Magna Carta, which in 1215 required the King to call a "common counsel" (now called Parliament) to represent people, to hold courts in a fixed place, to guarantee fair trials, to guarantee free movement of people, to free the church from the state, and to guarantee rights of "common" people to use the land.[5] (Most of Magna Carta is no longer in force; those principles it established that still exist are mostly protected by other enactments.) After the Wars of the Three Kingdoms and the Glorious Revolution, the Bill of Rights 1689 and the Claim of Right Act 1689 cemented Parliament's position as the supreme law-making body, and said that the "election of members of Parliament ought to be free".

The Treaty of Union in 1706, followed by the Acts of Union 1707 (one by each national parliament) unified the Kingdoms of England (which incorporated Wales) and Scotland. Ireland joined in a similar way through the Acts of Union 1801. The Irish Free State separated after the 1921 Anglo-Irish Treaty took effect in 1922. Northern Ireland remained within the union.

Tekst konstytucji Wielkiej Brytanii przedstawiono na rysunku 8, wykorzystując metodę fragmentacji opartą na NLP. Ponieważ NLP rozumie tekst i kontekst, dzieli go na odpowiednim poziomie, stosując odpowiednie tokeny, aby zachować sens i dokładność.

The constitution of the United Kingdom or British constitution comprises the written and unwritten arrangements that establish the United Kingdom of Great Britain and Northern Ireland as a political body. Unlike in most countries, no attempt has been made to codify such arrangements into a single document, thus it is known as an uncodified constitution. This enables the constitution to be easily changed as no provisions are formally entrenched;[2] the Supreme Court of the United Kingdom recognises that there are constitutional principles, including parliamentary sovereignty, the rule of law, democracy, and upholding international law.[3]

The Supreme Court also recognises that some Acts of Parliament have special constitutional status, and are therefore part of the constitution.[4] These include Magna Carta, which in 1215 required the King to call a "common counsel" (now called Parliament) to represent people, to hold courts in a fixed place, to guarantee fair trials, to guarantee free movement of people, to free the church from the state, and to guarantee rights of "common" people to use the land.[5] (Most of Magna Carta is no longer in force; those principles it established that still exist are mostly protected by other enactments.) After the Wars of the Three Kingdoms and the Glorious Revolution, the Bill of Rights 1689 and the Claim of Right Act 1689 cemented Parliament's position as the supreme law-making body, and said that the "election of members of Parliament ought to be free".

The Treaty of Union in 1706, followed by the Acts of Union 1707 (one by each national parliament) unified the Kingdoms of England (which incorporated Wales) and Scotland. Ireland joined in a similar way through the Acts of Union 1801. The Irish Free State separated after the 1921 Anglo-Irish Treaty took effect in 1922. Northern Ireland remained within the union.

Te strategie dzielenia na fragmenty są przydatne i ważne dla każdego wybranego dostawcy lub LLM. W kolejnych sekcjach pokażemy, jak stosować te strategie. Zaczniemy od Sentence Splitter, narzędzia do dzielenia tekstu, które dzieli tekst na podstawie nowego wiersza.

Obsługa nieznanej złożoności

Czasami nie znamy z góry złożoności i długości zapytań użytkowników. W takich przypadkach implementacje RAG, które radzą sobie z nieznaną długością i złożonością zapytań użytkowników, mogą być trudne. Oto kilka strategii określania podejścia do dzielenia na fragmenty w takich scenariuszach:

* Adaptacyjne dzielenie na fragmenty — zaimplementuj adaptacyjny mechanizm dzielenia na fragmenty, który automatycznie dostosowuje rozmiar fragmentów w zależności od długości i złożoności zapytania. Mniejsze fragmenty można wykorzystać do krótszych, prostszych zapytań, natomiast większe fragmenty mogą być potrzebne do uchwycenia niezbędnego kontekstu w przypadku dłuższych, bardziej złożonych zapytań.

* Heurystyka wstępnego przetwarzania — użyj heurystyk do analizy zapytania przed jego fragmentacją. Heurystyki te mogą oszacować złożoność, analizując takie czynniki, jak liczba unikalnych słów, obecność specjalistycznej terminologii lub struktura składniowa. Na podstawie tej oceny mechanizm fragmentacji może dostosować rozmiar fragmentów.

* Dynamiczne okno wyszukiwania — zaimplementuj dynamiczne okno wyszukiwania, które rozszerza się lub zwęża w zależności od zapytania. Jeśli początkowe wyniki wyszukiwania są niezadowalające, okno można dostosować, aby uwzględnić więcej lub mniej dokumentów lub zmienić szczegółowość fragmentacji.

* Nakładające się fragmenty — twórz nakładające się fragmenty, aby zapewnić, że żadne krytyczne informacje nie zostaną utracone na granicach fragmentów. Takie podejście może pomóc w zachowaniu kontekstu, gdy zapytania obejmują wiele fragmentów. W zależności od przypadku użycia może to

również przyćmić inne informacje, czego nie powinno się robić domyślnie. * Podejścia ML — Użyj tradycyjnych modeli ML do przewidywania optymalnego rozmiaru fragmentu na podstawie charakterystyki zapytania. Model można trenować na zbiorze danych zapytań i optymalnych rozmiarach fragmentów określonych na podstawie wydajności zestawu walidacyjnego.

* Strategie awaryjne — Wprowadź strategie awaryjne na wypadek, gdy początkowe fragmentowanie nie przyniesie dobrych rezultatów. Może to obejmować ponowne zapytanie z różnymi rozmiarami fragmentów lub zastosowanie różnych strategii fragmentowania, jeśli początkowa odpowiedź nie spełnia określonych progów ufności.

* Pętla sprzężenia zwrotnego — Zaimplementuj pętlę sprzężenia zwrotnego, w której interakcje użytkownika mogą pomóc w dostosowaniu fragmentowania. Jeśli użytkownik wskaże niezadowalającą odpowiedź, system może automatycznie wypróbować różne strategie fragmentowania, aby ją poprawić.

* Podejścia hybrydowe — Połącz kilka powyższych strategii, aby obsługiwać różne zapytania. Na przykład adaptacyjne fragmentowanie ze strategią awaryjną, która stale wykorzystuje opinie użytkownika, może usprawnić mechanizm fragmentowania.

W praktyce optymalnym rozwiązaniem byłoby połączenie tych strategii w konkretnym przypadku użycia, a do zwiększenia wydajności potrzebna jest metoda prób i błędów. Co więcej, elastyczność komponentów systemu umożliwia wprowadzanie zmian i ulepszeń w mechanizmie dzielenia na fragmenty, w miarę gromadzenia coraz większej ilości informacji o rodzajach zapytań wprowadzanych przez użytkowników.

Dzielenie zdań

Rozdzielacz oparty na zdaniach to metoda, która dzieli tekst na fragmenty w oparciu o granice zdań, takie jak kropki, znaki zapytania lub wykrzykniki. Metoda ta pozwala zachować sens i spójność tekstu, ponieważ każdy fragment zawiera jedno lub więcej pełnych zdań. Listing 1 przedstawia prostą implementację: tekst przychodzący jest dzielony na zdania za pomocą wyrażeń regularnych. Funkcja dzieli tekst wejściowy przy każdym wystąpieniu kropki (.), wykrzyknika (!) lub znaku zapytania (?). Znaki te są zazwyczaj używane do oznaczenia końca zdania w języku angielskim. Rezultatem jest lista ciągów znaków, z których każdy stanowi zdanie z tekstu oryginalnego.

Listing 1: Funkcja zdania podzielonego

```
def split_sentences(text):
    sentences = re.split('[.!?]', text)
    sentences = [sentence.strip() for sentence in sentences if sentence]
    return sentences
```

Splits the sentence at every occurrence of these characters

Innym sposobem na zaimplementowanie tego samego efektu jest użycie dzielnika opartego na zdaniach, takiego jak biblioteka textwrap w Pythonie. Ta funkcja, wrap(), może podzielić ciąg znaków na listę ciągów znaków na podstawie zadanej szerokości. Możemy przekazać dodatkowe parametry, aby upewnić się, że słowa nie zostaną rozdzielone w środku zdania.

Listing 2: Dzielenie zdań za pomocą zawijania tekstu

```
def split_sentences_by_textwrap(text):
    max_chunk_size = 2048          <- Sets the maximum chunk size

    chunks = textwrap.wrap(text,   <- Splits the text into chunks
        width=max_chunk_size,
        break_long_words=False,
        break_on_hyphens=False)

    return chunks
```

Należy zauważyć, że funkcje textwrap.wrap() i re.split() służą różnym celom, a ich wydajność, szybkość i dokładność zależą od konkretnego przypadku użycia. Pierwotnym celem biblioteki textwrap jest wyświetlanie tekstu oraz pomoc w formatowaniu i zawijaniu ciągów znaków, w których chcemy kontrolować maksymalną długość wiersza. Jest ona wydajna i szybka w zamierzonym zastosowaniu. Nie została jednak zaprojektowana do dzielenia tekstu na zdania, więc jej użycie w tym celu może nie być dokładne. Na przykład, może podzielić zdanie w środku, jeśli jest ono dłuższe niż określona szerokość. Funkcja split() w wyrażeniach regularnych dzieli ciąg znaków tam, gdzie pasuje wzorzec. Potrafi dobrze podzielić tekst na zdania, gdy jest używana ze wzorcem takim jak „[.!?]". Jest również szybka i skuteczna w tym, co robi. Nie uwzględnia jednak długości wiersza ani granic słów, więc jeśli trzeba ograniczyć długość każdego fragmentu, re.split() nie będzie najlepszym rozwiązaniem. Pod względem szybkości obie funkcje działają dość szybko i powinny działać dobrze w większości typowych zastosowań. Prędkość mogłaby stanowić problem w przypadku bardzo długich ciągów znaków, ale w większości przypadków różnica nie byłaby zauważalna. Jeśli chodzi o dokładność, jeśli potrzebujemy podzielić tekst na zdania, funkcja re.split() byłaby dokładniejsza. Jeśli potrzebujemy zawinąć tekst do określonej długości wiersza, funkcja textwrap.wrap() byłaby dokładniejsza.

Obie funkcje są dość wydajne, ponieważ są częścią standardowej biblioteki Pythona i zaimplementowane w języku C. Wydajność zależy również od rozmiaru i złożoności ciągu wejściowego.

Podział na fragmenty z wykorzystaniem przetwarzania języka naturalnego

Jak opisano we wcześniejszym przykładzie, możemy wykorzystać podejście przetwarzania języka naturalnego (NLP) do podziału tekstu na fragmenty; fragmenty te mogą być oparte na cechach językowych, takich jak zdania, frazy lub encje. W porównaniu z opisanymi wcześniej metodami dzielenia zdań, ta metoda pozwala uchwycić znaczenie i kontekst tekstu, ale może wymagać większych zasobów obliczeniowych i wiedzy dziedzinowej. Przyjrzyjmy się kilku przykładom z wykorzystaniem dwóch najpopularniejszych obecnie bibliotek NLP — Natural Language Toolkit (NLTK) i spaCy.

KORZYSTANIE Z NLTK

NLTK to jedna z najbardziej znanych bibliotek do przetwarzania języka naturalnego i analizy tekstu. Zapewnia łatwy w użyciu interfejs do wielu korpusów i zasobów leksykalnych. Ponadto zawiera zestaw bibliotek do przetwarzania tekstu, takich jak klasyfikacja, tokenizacja, stemming, tagowanie, parsowanie i wiele innych. NTLK można zainstalować na wiele sposobów; w przypadku condy możemy użyć następującego polecenia: conda install -c anaconda nltk. W przypadku pip możemy użyć polecenia pip install nltk. Zanim będziemy mogli korzystać z NLTK, musimy zainstalować dane NLTK, co można zrobić za pomocą programu do pobierania danych NLTK. Prostym sposobem na to jest uruchomienie interpretera Pythona z uprawnieniami administratora i wykonanie poniższych poleceń. Więcej informacji można znaleźć na stronie https://www.nltk.org/data.html:

```
>>> import nltk
```

```
>>> nltk.download()
```

Poniższa lista pokazuje, jak zaimplementować NLTK za pomocą funkcji sent_tokenize() w celu podzielenia tekstu na zdania

Listing 3: Podział tekstu na fragmenty za pomocą NLP

```
def split_sentences_by_nltk(text):

chunks = []

for sentence in nltk.sent_tokenize(text):

chunks.append(sentence)

return chunks
```

Funkcja sent_tokenize() korzysta z instancji PunktSentenceTokenizer, nienadzorowanego tokenizera opartego na uczeniu maszynowym, który jest wstępnie wytrenowany i gotowy do dzielenia zdań. Jeśli tekst jest bardzo długi, warto rozważyć użycie wyrażenia generatora zamiast wyrażenia listy, aby zwiększyć wydajność pamięci.

Poniższy wykaz pokazuje, jak poprzednią funkcję można zapisać jako funkcję generatywną.

Listing 4: Podział na fragmenty z wykorzystaniem przetwarzania języka naturalnego: funkcja generatywna

```
def split_sentences_by_nltk(text):

for sentence in nltk.sent_tokenize(text):

yield sentence
```

NLTK może być bardzo przydatny w przypadku dzielenia tekstu na fragmenty. Potrafi wykrywać granice zdań i dzielić je w tych wierszach, a także skutecznie dzielić teksty na pojedyncze zdania, co może być przydatne w przypadku dzielenia dużych tekstów na fragmenty, zapewniając jednocześnie, że zdania nie zostaną podzielone w środku. Z perspektywy przedsiębiorstwa warto zauważyć, że chociaż NLTK jest wszechstronny i odpowiedni do celów badawczych i edukacyjnych, nie zawsze jest najwydajniejszy pod względem szybkości. Inne biblioteki, takie jak spaCy, mogą być bardziej odpowiednie dla aplikacji produkcyjnych, zwłaszcza przy przetwarzaniu dużych ilości tekstu.

KORZYSTANIE ZE spaCy

spaCy to darmowa, open source'owa biblioteka NLP dla Pythona, która oferuje szeroki zakres zadań NLP, w tym segmentację zdań, rozpoznawanie jednostek nazwanych, tagowanie części mowy i analizę zależności. Jest również przydatna do dzielenia tekstu na fragmenty i grupowania słów w sensowne jednostki, takie jak frazy rzeczownikowe, frazy czasownikowe i frazy przyimkowe.

spaCy to dobry wybór dla implementacji RAG, ponieważ jest wydajny i szybki, szczególnie podczas przetwarzania dużych ilości tekstu w czasie rzeczywistym. Jest dokładny i niezawodny oraz można go dostosować do konkretnych potrzeb. Na przykład, spaCy może być używany do dzielenia tekstu na części za pomocą różnych teorii lingwistycznych, takich jak gramatyka struktur frazowych i gramatyka zależności. Zanim będziemy mogli używać spaCy, musimy zainstalować pakiety i pobrać odpowiedni, wstępnie wytrenowany model języka, z którego będzie korzystać spaCy. Jeśli używamy conda, możemy zainstalować spaCy za pomocą conda install -c conda-forge spacy. Jeśli używamy pip, możemy użyć następującego polecenia: pip install spacy. W tym przykładzie pobieramy mały, uniwersalny model

języka angielskiego o nazwie en_core_web_sm za pomocą następującego polecenia: python -m spacy download en_core_web_sm. spaCy oferuje dodatkowe modele dla różnych celów i języków. Oprócz małego modelu angielskiego dostępne są modele średni i duży — odpowiednio en_core_web_md i en_core_web_lg, dla bardziej kompleksowych wektorów słów. Im większy model, tym dłuższy czas przetwarzania. Wybór modelu wiąże się z czymś więcej niż tylko rozmiarem; należy uwzględnić dokładność, języki i domenę. Więcej informacji na temat wstępnie wytrenowanych modeli można znaleźć na stronie https://spacy.io/usage/models/. Poniższa lista pokazuje, jak można wykorzystać spaCy do dzielenia na fragmenty. W tym przykładzie uwzględniamy liczbę tokenów dla okien kontekstowych LLM, a także mamy możliwość nakładania tekstu między fragmentami, aby zapewnić ciągłość kontekstu.

Listing 5: Dzielenie zdań na części za pomocą spaCy

```
def split_sentences_by_spacy(text, max_tokens, overlap=0,
     model="en_core_web_sm"):
    nlp = spacy.load(model)          <- Loads the spaCy model

    doc = nlp(text)                  <- Tokenizes the text into
    sentences = [sent.text for sent in doc.sents]   sentences using spaCy

    tokens_lengths = [count_tokens(sent) for sent in sentences]   <- Tokenizes sentences and
                                                                     accumulates tokens
    chunks = []
    start_idx = 0

    while start_idx < len(sentences):
        current_chunk = []
        current_token_count = 0
        for idx in range(start_idx, len(sentences)):
            if current_token_count + tokens_lengths[idx] > max_tokens:
                break
            current_chunk.append(sentences[idx])
            current_token_count += tokens_lengths[idx]

        chunks.append(" ".join(current_chunk))
                                     Sliding window
        if overlap >= len(current_chunk):   <- adjustment
            start_idx += 1
        else:
            start_idx += len(current_chunk) - overlap

    return chunks
```

Te techniki mają różne korzyści i charakterystykę obliczeniową. Wypróbujmy je wszystkie i porównajmy ich wydajność, czas trwania i efekt. Na przykład używamy Azure OpenAI i danych z Mistrzostw Świata Kobiet FIFA 2023 . Miało to miejsce w 2023 roku i w momencie publikacji tej publikacji LLM nie posiadał tej wiedzy, ponieważ wykraczało to poza zakres szkolenia. W tym przykładzie zapisujemy stronę Wikipedii dotyczącą Mistrzostw Świata Kobiet FIFA 2023 jako surowe pole tekstowe. Ten plik nie jest przetwarzany, a wynikowy plik jest na tyle chaotyczny, że odzwierciedla wiele rzeczywistych problemów, z którymi borykają się przedsiębiorstwa. W tym przykładzie, jak pokazano na listingu 6, analizujemy cztery różne techniki fragmentowania, używając tego samego pliku i opisujemy czas potrzebny na wykonanie każdej techniki, liczbę utworzonych fragmentów i użyte tokeny. Używamy również GPT3 do utworzenia podsumowania odczytanego tekstu. Zaczynamy od załadowania zapisanego pliku tekstowego o nazwie women_fifa_worldcup_2023.txt. Stosujemy cztery

różne techniki fragmentacji oddzielnie, a następnie używamy tego samego trybu GPT do ich podsumowania. Najpierw fragmentujemy, używając podstawowej metody fragmentacji zdań, i przetwarzamy je. Następnie przetwarzamy ten sam plik, używając textwrap, NLTK i spaCy. Rejestrujemy proste dane telemetryczne przy każdym uruchomieniu i wyświetlamy je wszystkie na końcu, wraz z podsumowaniem. Warto zauważyć, że kilka funkcji pomocniczych, takich jak get_embedding(), count_tokens() i tak dalej, zostało już wcześniej wykorzystanych w książce – nie przywołujemy ich ponownie dla zwięzłości.

Listing 6: Porównanie fragmentów zdań

```
import nltk
import spacy
...

GPT_MODEL = "gpt-35-turbo"

def generate_summaries(chunks):
    summaries = []                  ◁── Empty list to store the summaries
    # loop through each chunk
    for chunk in tqdm(chunks):
        prompt = f"Summarize the following text in one
        ➥sentence:\n{chunk}\nSummary:"
        response = openai.Completion.create(    ◁── Completion to generate a summary for the chunk
            engine=GPT_MODEL,
            prompt=prompt,
            max_tokens=800,
            temperature=0.7        )
        summary = response.choices[0].text
        summaries.append(summary)
        sleep(1)                    ◁── Rate limiting

    # return the list of summaries
    return summaries

def process_chunks(sentences):
    sentence_embeddings = []
    total_token_count = 0

    for i, sentence in enumerate(tqdm(sentences)):
        total_token_count += count_tokens(
              ➥sentence, "cl100k_base")    ◁── Counts tokens in the sentence
        embedding = get_embedding(sentence)
        sentence_embeddings.append([sentence, embedding])

    print("\tNumber of sentence embeddings:", len(sentence_embeddings))
    print("\tTotal number of tokens:", total_token_count)

    return sentence_embeddings

TEXT_FILE = f"data/women_fifa_worldcup_2023.txt"    ◁── File that we want to chunk

with open(TEXT_FILE, "r") as f:
    text = f.read()

print("1. Simple sentence chunking ...")
sentences = split_sentences(text)
```

```
process_chunks(sentences)

print("="*20)
# ==================================

#Reset variables
summaries = []
sentences = []
sentence_embeddings = []
total_token_count = 0
chunks = []

print("2. Sentence chunking using textwrap ...")
chunks = split_sentences_by_textwrap(text)      <- Chunks text using textwrap
process_chunks(chunks)

# ==================================

#Reset variables
...

print("3. Sentence chunking using NLTK ...")
chunks = split_sentences_by_nltk(text)      <- Chunks text using NLTK
process_chunks(chunks)

# ==================================

#Reset variables
...

print("4. Sentence chunking using spaCy ...")
chunks = split_sentences_by_spacy(text,
               ➥max_tokens=2000, overlap=0)      <- Chunks text using spaCy
process_chunks(chunks)

# ==================================
summaries = generate_summaries(chunks)      <- Generates summaries for each chunk using OpenAI API
print("Summaries generated by OpenAI API:")
print(summaries)
```

Tabela 4 przedstawia wynik po uruchomieniu tej funkcji, wraz z czasem trwania w sekundach. Zgodnie z oczekiwaniami, czas potrzebny na przetworzenie tego samego tekstu wejściowego różni się znacznie w zależności od użytej techniki.

Metoda kawałkowania	Liczą się osadzania	Liczą się tokeny	Czas wykonania (sek.)
Prosty	120	5815	16,96
Korzystanie z zawijania tekstu	12	5933	1,66
Korzystanie z NLTK	105	5909	13.31
Używanie spacji	4	5876	5.8

Poniżej znajduje się podsumowanie wygenerowane przez LLM z wykorzystaniem fragmentów spaCy; podsumowania te są zwięzłe i pouczające, co było naszym zamierzeniem:

Podsumowania wygenerowane przez API OpenAI:

[„Mistrzostwa Świata Kobiet FIFA to międzynarodowe rozgrywki piłkarskie, w których biorą udział seniorskie żeńskie reprezentacje narodowe członków FIFA. Odbywają się co cztery lata od 1991 roku; najbardziej utytułowaną drużyną są Stany Zjednoczone, z czterema tytułami mistrzowskimi…”]

Ten przykład pokazuje, że podejście z zawijaniem tekstu jest najszybsze i zajmuje 1,66 sekundy; nie oznacza to jednak, że podejście z zawijaniem tekstu jest zawsze najodpowiedniejsze i że powinniśmy je zastosować. Musimy to ocenić dla każdej sytuacji, w zależności od rodzaju informacji i przypadku użycia. Przyjrzyjmy się czynnikom decyzyjnym wymaganym do wyboru najlepszej strategii fragmentowania.

WYBÓR WŁAŚCIWEJ STRATEGII

To, czy zastosować strategię fragmentacji opartą na NLP, czy podejście fragmentacji o stałej długości, zależy od konkretnych wymagań i ograniczeń danego zadania. Tabela 5 przedstawia niektóre czynniki decyzyjne.

Czynnik decyzyjny	Opis
Wymagania zadania	Jeśli zadanie wymaga zrozumienia niuansów językowych, np. odpowiadania na pytania zależne od kontekstu lub generowania spójnego tekstu, lepszym rozwiązaniem jest dzielenie tekstu na fragmenty w oparciu o NLP.
Wydajność	Jeśli zachowanie kontekstu nie jest krytyczne i występują ograniczenia wydajnościowe, lepszym wyborem może być dzielenie danych na fragmenty o stałej długości.
Dostępność zasobów	Podział na fragmenty o stałej długości wymaga mniej zasobów w przypadku projektów o ograniczonych zasobach obliczeniowych i jest łatwiejszy do skalowania.
Charakterystyka danych	Podział na fragmenty oparty na NLP może wykorzystywać te granice w przypadku tekstów o wyraźnych rozgraniczeniach językowych (takich jak dokumenty o dobrej strukturze). Natomiast podział na fragmenty o stałej długości może być bardziej praktyczny, jeśli tekst jest słabo ustrukturyzowany lub granice są niejasne.

Można zacząć od metody o stałej długości, ponieważ jest łatwa, a następnie przejść na metodę opartą na NLP, gdy wymagana jest większa złożoność. Niektóre zaawansowane systemy mogą nawet wykorzystywać obie metody, stosując fragmentację o stałej długości do szybkiego przetwarzania dużych ilości tekstu, a następnie fragmentację opartą na NLP do mniejszych, bardziej kontrolowanych fragmentów, aby poprawić kontekst i znaczenie. Zmieńmy tematy i zobaczmy, jak możemy fragmentować inne dokumenty, takie jak pliki PDF.

Podział plików PDF na fragmenty

Na poziomie ogólnym podział plików PDF na fragmenty jest dość podobny do podziału zdań na fragmenty. Istnieją różne opcje dla plików PDF, które nie są zbyt skomplikowane i zawierają podstawowe tabele lub obrazy. Prostą metodą na początek jest użycie biblioteki PyPDF2. PyPDF2 to biblioteka PDF w Pythonie o otwartym kodzie źródłowym, która umożliwia wykonywanie różnych operacji na stronach PDF, takich jak dzielenie, scalanie, kadrowanie i przekształcanie. Umożliwia również wyodrębnianie tekstu, danych niestandardowych, haseł i metadanych z plików PDF. Listing 7.7 pokazuje, jak z niej korzystać. Możemy zainstalować PyPDF2 za pomocą następującego polecenia w conda: conda install -c conda-forge pypdf2 lub, jeśli używasz pip, pip install pypdf2. Wynik to tekst, który można podzielić na fragmenty i przetworzyć jak każdy inny tekst omówiony wcześniej. Ta biblioteka nie obsługuje obrazów; jeśli w pliku PDF znajdują się jakiekolwiek obrazy, zostaną one zignorowane. Należy pamiętać, że poniższy listing zawiera tylko odpowiednią sekcję dla zwięzłości; .

Listing 7: Wyodrębnianie tekstu z pliku PDF

import PyPDF2

def extract_text_from_pdf(pdf_path):

with open(pdf_path, 'rb') as file:

reader = PyPDF2.PdfReader(file)

print("Number of PDF pages:", len(reader.pages))

text = ""

for page in reader.pages:

page_text = page.extract_text()

text += page_text

#print(page_text)

return text

POSTĘPOWANIE Z TABELAMI I OBRAZAMI W PLIKACH PDF

Chociaż obsługa tekstu w ostatnim przykładzie wydaje się dość prosta, pliki PDF mogą znacznie komplikować ten proces. Dokładność ekstrakcji tekstu zależy od samego pliku PDF, ponieważ nie wszystkie pliki PDF kodują tekst w sposób umożliwiający jego łatwą ekstrakcję. Poniższa lista przedstawia przykład przetwarzania obrazów i tabel w celu wyodrębnienia fragmentów z plików PDF, ale ogólnie rzecz biorąc, będzie to trudne.

Listing 8: Przykład wyodrębniania tabel i obrazów

```
from PIL import Image
import tabula
from pdfminer.high_level import extract_pages
from PyPDF2 import PdfReader
...

# Define the PDF file path
pdf_file = f"data/test.pdf"
output_folder = f"data/temp/"

text = ''      <-- Creates an empty string to store the text

def process_element(element, iw):
    global text
    if isinstance(element, LTTextBox):
        text += element.get_text()
    elif isinstance(element, LTImage):
```

```
            bmp_file = iw.export_image(element)
            bmp_file = os.path.join(iw.outdir, bmp_file)
            img = Image.open(bmp_file)
            png_file = bmp_file.rsplit('.', 1)[0] + '.png'
            img.save(png_file)
        if isinstance(element, LTFigure):
            for child in element:
                process_element(child, iw)

iw = ImageWriter(output_folder)

page = next(extract_pages(pdf_file))

for element in page:
    process_element(element, iw)

with open(output_folder + 'text.txt', 'w', encoding='utf-8') as f:
    f.write(text)

tables = tabula.read_pdf(pdf_file, pages='all')

for i, table in enumerate(tables):
    table.to_csv(f'{output_folder}table_{i}.csv', index=False)
```

Opens the BMP file with PIL

Exports the image as a BMP file

Converts the image to PNG and saves

Creates an ImageWriter object to save the images

Gets the first page from the PDF file

Reads the tables

Saves each table into a separate file

Obrazy i tabele są trudne do analizy i obsługi w plikach PDF, szczególnie w sposób spójny i przewidywalny. Jednym ze sposobów rozwiązania tego problemu przez przedsiębiorstwa jest szeregowe łączenie innych modeli uczenia maszynowego (ML) zamiast prób samodzielnego analizowania dokumentów. Usługa Azure Document Intelligence firmy Microsoft umożliwia to.

AZURE DOCUMENT INTELLIGENCE

Azure AI Document Intelligence to usługa w chmurze, która wykorzystuje zaawansowane modele uczenia maszynowego (ML) do automatycznego wyodrębniania tekstu, wartości kluczowych, tabel i struktur z dokumentów, przekształcając je w użyteczne dane. Oferuje trzy typy modeli uczenia maszynowego (ML) do analizy dokumentów: modele predefiniowane dla typowych scenariuszy (np. identyfikatory, paragony), modele niestandardowe wytrenowane na podstawie danych użytkownika oraz modele analizy dokumentów do wyodrębniania ustrukturyzowanej treści. W przeciwieństwie do wielu pakietów Pythona do obsługi plików PDF, obsługuje wiele formatów dokumentów, obsługuje złożone układy, tekst odręczny i obiekty, a także umożliwia tworzenie niestandardowych modeli uczenia maszynowego. Integracja jest prosta za pośrednictwem interfejsu API REST, który umożliwia wyodrębnianie danych z dokumentów, takich jak pliki PDF, i wykorzystanie wzorca RAG do podsumowania lub generowania odpowiedzi. Listing 9 pokazuje, jak analizować dokument za pomocą gotowego modelu układu. Rozpoczynamy analizę przykładowego dokumentu, korzystając z gotowego modelu układu, i iterujemy wynik. Wykrywa on tekst i tabele dla każdej strony, w tym wiersze i kolumny. Należy pamiętać, że Document Intelligence nie jest jeszcze dostępny w systemie Conda, więc użyjemy pip, aby go zainstalować: pip install azure-ai-formrecognizer

Listing 9: Wstępnie zbudowane operacje układu w usłudze Azure AI Document Intelligence

```
from azure.core.credentials import AzureKeyCredential
from azure.ai.formrecognizer import DocumentAnalysisClient
```

```
endpoint = "YOUR_FORM_RECOGNIZER_ENDPOINT"
key = "YOUR_FORM_RECOGNIZER_KEY"

# sample document
pdf_file = f"YOUR_PDF_FILE"

document_analysis_client = DocumentAnalysisClient(
    endpoint=endpoint, credential=AzureKeyCredential(key)
)

poller = poller = document_analysis_client.begin_analyze_document_from_url(
    "prebuilt-layout", pdf_file)
result = poller.result()

for page in result.pages:
    for line_idx, line in enumerate(page.lines):
        print(
         "...Line # {} has text content '{}'".format(
        line_idx,
        line.content.encode("utf-8")
        )
    )

    for selection_mark in page.selection_marks:
        print("...Selection mark is '{}'".format(
         selection_mark.state,
         )
    )

for table_idx, table in enumerate(result.tables):
    print(
        "Table # {} has {} rows and {} columns".format(
        table_idx, table.row_count, table.column_count
        )
    )

    for cell in the table.cells:
        print(
            "...Cell[{}][{}] has content '{}'".format(
            cell.row_index,
            cell.column_index,
            cell.content.encode("utf-8"),
            )
        )
```

Calls the API to analyze a PDF using a pre-build layout

Gets the results from the analysis

Iterates through all the pages in the PDF

Extracts the text on each line on the page

Examines whether there is a Selection mark

Parses tables found in the PDF

Podsumowanie

* LLM-y nie posiadają żadnych aktualnych informacji po zakończeniu szkolenia i nie znają informacji prywatnych ani zastrzeżonych. Generowanie wspomagane wyszukiwaniem (RAG) to technika, która pomaga pokonać te ograniczenia.

* RAG to potężna technika, która dostarcza aktualnych i ugruntowanych informacji, prowadząc do lepszych odpowiedzi LLM. Pomaga również ugruntowywać dane i może poprawić generowanie LLM pod względem jakości, różnorodności i personalizacji.

* Wyszukiwania wektorowe oparte na indeksach wektorowych i bazach danych odgrywają kluczową rolę w zwiększeniu wydajności i skalowalności komponentu wyszukiwania w implementacjach RAG. Umożliwiają one wyszukiwanie semantyczne w czasie rzeczywistym na dużą skalę, co jest niezbędne w aplikacjach wymagających szybkiego dostępu do ogromnych ilości informacji.

* RAG musi stawić czoła różnym wyzwaniom, a fragmentacja jest jednym z najważniejszych. Ze względu na ograniczenia okna zawartości LLM, musimy fragmentować duży korpus danych i stosować różne techniki podziału informacji — o stałej długości, przesuwanym oknie, oparte na interpunkcji, oparte na

sekcjach lub adaptacyjne. Każda z nich ma swoje zalety i wady i należy ją rozpatrywać w kontekście przypadku użycia, kształtu i typu danych.

* Zrozumienie i parsowanie plików PDF na fragmenty jest trudne, zwłaszcza jeśli zawierają one obrazy i tabele. Łańcuchowe łączenie innych modeli uczenia maszynowego, takich jak Azure Document Intelligence, może to uprościć.

* W połączeniu z szybką inżynierią, RAG pomaga sprostać takim wymaganiom, jak dynamiczny dostęp do informacji, opłacalność, uziemienie, cytowanie, skalowalność i dostosowanie do różnych scenariuszy i kontekstów przedsiębiorstwa.