



Lekcja [S03E02](#) pokazała nam, że bazy wektorowe nie wystarczają do skutecznego przeszukiwania danych. Co prawda wsparcie ze strony modelu poprawia sytuację, ale nie rozwiązuje wszystkich problemów. Pierwszym z brzegu przykładem może być poszukiwanie akronimów, numerów serii czy zamówienia, lub wyrażeń, których model nie potrafi opisać w embeddingu.

Zatem gdy mamy gdy mamy do czynienia ze słowami kluczowymi i precyzyjnym dopasowaniem, bazy wektorowe okazują się niewystarczające, a wyszukiwanie semantyczne musi zostać uzupełnione wyszukiwaniem pełnotekstowym.

Łączenie różnych technik wyszukiwania określamy mianem wyszukiwania hybrydowego. Może ono przybierać różne formy i konfiguracje w zależności od potrzeb. Przykładowo, czasami wystarczy użyć PostgreSQL z pgvector i pgsearch do przechowywania danych oraz do wyszukiwania semantycznego i pełnotekstowego, co powinno wystarczyć nam na potrzeby małych projektów. Innym razem będzie nam zależało na rozdzieleniu tych odpowiedzialności na różne narzędzia.

W tej lekcji zajmiemy się więc wyszukiwaniem hybrydowym, które umożliwia nam budowanie zaawansowanych systemów Retrieval-Augmented Generation, w tym także pamięci dla agenta AI.

Klasyczne silniki wyszukiwania

Silniki wyszukiwania, takie jak Algolia czy ElasticSearch, umożliwiają zaawansowane przeszukiwanie z uwzględnieniem dopasowania pełnotekstowego (full-text search), dopasowania 'rozmytego' (fuzzy search),

zawężania obszaru przeszukiwania (faceted search), filtrowania oraz oceny zwróconych rezultatów.

Podstawowa interakcja odbywa się na podobnej zasadzie jak w przypadku baz wektorowych. Mamy więc indeks, a w nim dokumenty posiadające swoje właściwości. Można to zobaczyć, zakładając konto na algolia.com, z którego możemy pobrać identyfikator aplikacji oraz klucz API (administrator). Następnie, po uruchomieniu przykładu algolia, na naszym koncie utworzony zostanie indeks `dev_comments` i pojawią się w nim trzy rekordy.

The screenshot shows the Algolia Index interface. At the top, it displays 'Application' (Alice) and 'Index' (dev_comments). Below this, the 'Index' section shows 3 records, 0 events, and an average record size of 138.33B. A message encourages enhancing search relevance by sending event data. There are buttons for 'New...', 'Add records', and 'Manage index'. The 'Browse' tab is selected, followed by 'Configuration', 'Replicas', 'Search API Logs', 'Stats', and 'UI Demos'. A search bar at the top right shows '3 hits matched in 1ms'. The main area lists three documents:

objectID	author	text
"6993c6de-b522-4735-9714-8a207ceaae09"	"Adam"	"I believe in writing clean, maintainable code. Refactoring should be a regular part of our development process."
"5eb5b1c4-9b58-494c-acfe-3c2649308a0c"	"Mateusz"	"Optimizing our CI/CD pipeline could greatly e..."

At the bottom right of the document list, there are icons for edit, delete, and copy. The footer includes links for 'Last 7 days', 'n/a Searches', 'Configure for more analytics', and search parameters.

No i testowe zapytanie zwraca dopasowanie do dwóch z trzech dokumentów.

The terminal window shows a command being run: `aidev3 git:(main) ✘ bun algolia/app.ts`. The response indicates that the index already exists and data addition is skipped. Below this, a table displays the search results:

	Author	Text	ObjectID	MatchLevel	MatchedWords	UserScore
0	Adam	I believe in writing clean, maintainable code...	6993c6de-b522-4735-9714-8a207ceaae09	full	code	2
1	Mateusz	Optimizing our CI/CD pipeline could greatly e...	5eb5b1c4-9b58-494c-acfe-3c2649308a0c	full	code	1

Jeśli chcielibyśmy przeszukiwać wpisy tylko jednego z autorów, to w pierwszej kolejności w ustawieniach indeksu potrzebujemy wskazać pola, względem których będziemy chcieli filtrować. W naszym przypadku będzie to pole `author`.

The screenshot shows the 'Configuration' tab in the Algolia UI. Under 'Facets', there is a section for 'Attributes for faceting' where 'author' is listed. Other settings include 'Max values per facet' set to 100, 'Sort facet values by' set to 'Count', and 'Max facet hits' set to 10.

Tym razem wyszukiwanie zwraca nam tylko jeden wpis, dopasowany do autora wskazanego w filtrze.

The terminal output shows a search result for 'Adam'. The table contains one row with the following data:

#	Author	Text	ObjectID	MatchLevel	MatchedWords	UserScore
0	Adam	I believe in writing clean, maintainable code...	6993c6de-b522-4735-9714-8a207ceaae09	full	code	2

Mamy więc w tym momencie już zupełnie podstawy pracy z klasycznym silnikiem wyszukiwania i tym samym jesteśmy praktycznie gotowi na wdrożenie logiki dla wyszukiwania hybrydowego.

Synchronizacja danych

Już teraz widzimy, że w Algolia dokumenty mają swoje identyfikatory. To samo dotyczy Qdrant oraz bazy SQLite. Oznacza to, że wszystkie rekordy mogą być ze sobą powiązane, a całość opracowana tak, aby **baza danych stanowiła źródło prawdy**.

W przykładzie `sync` przygotowałem logikę odpowiedzialną za synchronizowanie informacji pomiędzy trzema źródłami: bazą danych, bazą wektorową i indeks silnika wyszukiwania. Dzięki temu jakiekolwiek zmiany wprowadzane w bazie danych, będą odzwierciedlane w pozostałych źródłach.

```

35  async function initializeData() {
36    const openAIService = new OpenAIService();
37    const algoliaService = new AlgoliaService(process.env.ALGOLIA_APP_ID!, process.env.ALGOLIA_API_KEY!);
38    const vectorService = new VectorService(openAIService);
39    const dbService = new DatabaseService('sync/database.db', algoliaService, vectorService);
40
41    const docs = await dbService.getAllDocuments();
42
43    if (docs.length === 0) {
44      for (const book of talebBooks) {
45        const document = {
46          uuid: uuidv4(),
47          name: book.title,
48          content: book.description,
49          source: "initialization",
50          conversation_uuid: uuidv4(),
51          type: "book",
52          description: `A book by ${book.author}`,
53          indexed: true
54        };
55
56        // Insert into SQLite database, which will sync to Algolia and Qdrant
57        await dbService.insertDocument(document);
58      }
59    }
60
61    await dbService.updateDocument('189ef4ad-041a-4ce9-933b-350c1c867080', { name: 'Antifragile 2' });
62    await dbService.deleteDocument('189ef4ad-041a-4ce9-933b-350c1c867080');
63
64    console.log("Initialization complete. Example data has been added to all stores.");
65  }
66}

```

W indeksie Algolia znajdują się wszystkie właściwości dokumentu, wliczając w to także unikatowy identyfikator.

objectID	"dde78d15-779d-45a8-87cf-79b1e275b3e0"
name	"Antifragile"
uuid	"dde78d15-779d-45a8-87cf-79b1e275b3e0"
content	"A book about things that gain from disorder and how to thrive in an uncertain world."
source	"initialization"
conversation_uuid	"80713c5d-fcc3-43f6-afeb-aa4c6d548e13"
type	"book"
description	"A book by Nassim Nicholas Taleb"
indexed	true

Podobnie wygląda to w Qdrant i tam również mamy komplet informacji, w tym także ten sam identyfikator.

```
Point cc653e1a-8e3a-432a-9121-c000a38ed73b

Payload:

text          The Black Swan: An exploration of rare and unpredictable events, and their massive impact on society and history.

metadata      ~ { 8 Items
                "uuid": "cc653e1a-8e3a-432a-9121-c000a38ed73b"
                "name": "The Black Swan"
                "content": "An exploration of rare and unpredictable events, a ..."
                "source": "initialization"
                "conversation_uuid": "0ea66188-908f-431b-a17e-1bce33251478"
                "type": "book"
                "description": "A book by Nassim Nicholas Taleb"
                "indexed": true
            }
```

No i ostatecznie mamy nasze źródło prawdy, czyli bazę danych SQLite na którym pracuje nasza aplikacja.

Search for field...	
id	integer
1	
name	text
The Black Swan	
content	text
An exploration of rare and unpredictable events, and their massive impact on society and history.	
source	text
initialization	
indexed	integer
1	
conversation_uuid	text
initial_books	
type	text
book	
description	text
A book by Nassim Nicholas Taleb	
created_at	text
2024-10-09 08:04:34	
updated_at	text
2024-10-09 08:04:34	
uuid	text
339e0a89-a4d2-42b8-9c61-21a6bd116119	

Zatem nawet w przypadku aplikacji działających na nieco większej skali, skorzystanie z trzech różnych narzędzi nie jest trudne, a daje nam duże możliwości, o czym już niebawem się przekonamy.

Także w temacie synchronizacji danych:

- Możemy skorzystać jedynie z PostgreSQL i dostępnych rozszerzeń, aby

przechowywać dane oraz przeszukiwać je na różne sposoby. Takie podejście może być wygodne, ale ograniczone. Tutaj można także rozważyć skorzystanie z Supabase.

- Możemy stworzyć **wspólny interfejs**, który pozwoli nam na automatyczne synchronizowanie danych pomiędzy różnymi źródłami wiedzy. Tutaj liczy się przede wszystkim utrzymanie **wspólnego identyfikatora**
- Nie zawsze synchronizacja będzie opierać się o dokładnie te same dane dla wszystkich źródeł. Już nawet w naszym przypadku, dane zapisane w Qdrant zawierają dodatkowo embedding, który nie występuje w pozostałych miejscach.

Wybór indeksowanych treści

Dość zasadne pytanie dotyczy tego, co przechowywać w oryginalnej bazie danych, a co w silnikach wyszukiwania. Odpowiedź zawsze będzie zależona od naszych indywidualnych potrzeb, ale możemy odpowiedzieć sobie na kilka ogólnych pytań, a także **uwzględnić zaangażowanie modelu w transformację treści**.

Zatem:

- **Jakich danych wymaga nasza aplikacja po stronie interfejsu** oraz w jaki sposób będziemy je wyświetlać? Odpowiedź na to pytanie determinuje strukturę informacji przechowywanych w bazie danych lub przynajmniej wskazuje kierunki.
- **Które z fragmentów danych będą wykorzystywane do przeszukiwania** oraz jak chcemy je filtrować lub ograniczać dostęp do nich? Odpowiedź na to pytanie wskazuje nam treść, która powinna trafić do silników wyszukiwania.
- **Jakich danych brakuje nam w oryginalnych dokumentach, a w ich utworzeniu może pomóc nam LLM?** Mowa tutaj głównie o klasyfikacji, tytułach, opisach, streszczeniach lub innej formie transformacji.
- **Czy duża liczba danych utrudni wyszukiwanie i jeśli tak, to co możemy z tym zrobić?** To wspierające pytanie pozwala zidentyfikować potrzebę dodatkowych filtrów i kategorii, które zostaną uwzględnione na etapie indeksowania oraz odzyskiwania / przeszukiwania
- **Które dane są stałe, a które tymczasowe?** Przykładowo w bazie mogą być zapisane wpisy tylko na potrzeby bieżącej konwersacji lub nawet jednego zadania, realizowanego w trakcie rozmowy. Musimy więc zadbać o możliwość ich łatwego wczytania, a także wykluczenia ich z głównych

wyników wyszukiwania o ile nie wchodzi to w konflikt z logiką aplikacji.

Refleksja nad powyższymi pytaniami pozwoli nam zarysować kształt struktury, którą możemy przełożyć na wszystkie źródła danych. Jakość tych odpowiedzi będzie także przekładać się na skuteczność działania całego systemu.

Wyszukiwanie hybrydowe

W przykładzie `hybrid` znajduje się rozszerzenie przykładu `sync`, jednak w tym przypadku na nieco innym zestawie danych przeprowadzamy przeszukiwanie dokumentów. Rzecznik w tym, że wykorzystujemy tutaj dwa silniki wyszukiwania, a więc rezultatem są **dwie listy dokumentów, które musimy ze sobą połączyć** korzystając z Rank Fusion. Konkretnie dokumenty otrzymują ranking liczony na podstawie sumy odwrotności ich pozycji z każdej listy. Chodzi dokładnie o ten fragment kodu:

```
76  return Array.from(resultMap.values())
77    .map((data) => ({
78      ...data,
79      score: (data.vectorRank ? 1 / (data.vectorRank) : 0) +
80           (data.algoliaRank ? 1 / (data.algoliaRank) : 0),
81    }))
82    .sort((a, b) => b.score - a.score);
83 }
```

Ujmując to prościej:

- Zapisujemy rezultaty obu wyszukiwań w formie **jednej listy**
- Liczymy dla nich `score` na podstawie wzoru `(1 / vector rank) + (1 / full text rank)`
- Sortujemy według tego rankingu

Oczywiście możemy zmieniać wagę dla poszczególnej listy, aby zmienić jej wpływ na ranking, zwiększając znaczenie wyszukiwania wektorowego lub pełnotekstowego.

Samo policzenie rankingu nie jest wszystkim, ponieważ gdy zajrzymy w kod, to przede wszystkim rzuci się w oczy fakt, że korzystamy z **dwoch zapytań**. Jedno zapisane językiem naturalnym jest kierowane do Qdrant, a drugie zawiera wyłącznie słowa kluczowe i jest kierowane do Algolia.

```

106
107 // Query for the vector search and full-text search
108 const QUERY = "Find me everything about people";
109 const DENSE_QUERY = "people";
110
111 // Determine the authors based on the query
112 const authors = await determineAuthors(QUERY);
113 const filter = buildFilter(authors);
114
115 // Vector search
116 const vectorResults = await performVectorSearch(QUERY, filter);
117 console.log("Vector results:");
118 vectorResults.forEach((result) => {
119   const author = result.author || "";
120   const textSnippet = result.content.slice(0, 75) || "";
121   const score = (result.score * 100).toFixed(2);
122   console.log(`[${author}]: ${textSnippet} (${score}%)`);
123 });
124
125 // Full-text search
126 const algoliaResults = await performAlgoliaSearch(DENSE_QUERY, { filters: buildAlgoliaFilter(authors) });
127 console.log("Algolia results:");
128 algoliaResults.forEach((hit) => {
129   console.log(hit.author + ': ' + hit.content.slice(0, 50));
130 });
131
132 // Calculate RRF scores
133 const rrfScores = calculateRRF(vectorResults, algoliaResults);

```

W testowym zestawie danych znajduje się lista książek Simona Sineka oraz Jima Collinса. Tylko trzy z nich bezpośrednio zawierają frazę "people", natomiast pośrednio ten temat porusza większość z nich.

Dla zapytania `Find me everything about people` Qdrant zwraca tytuły w których opisach pojawiła się fraza `people` na miejscach 1, 2 i 5. Z kolei Algolia zwraca tylko trzy rekordy, ale słowo `people` pada w każdym z nich.

```

QDRANT v0.15.0 - 2023-05-10T14:53:00Z
Fetching all documents
Found 13 documents
Vector results:
Jim Collins: Good to Great: "First who, then what. Get the right people on the bus, the (18.54%)
Simon Sinek: Start with Why: "People don't buy what you do; they buy why you do it. And (16.26%)
Simon Sinek: Leaders Eat Last: "The true price of leadership is the willingness to place (7.91%)
Simon Sinek: Leaders Eat Last: "Understanding the importance of OKRs and fostering psych (7.60%)
Jim Collins: Turning the Flywheel: "Disciplined people, thought, and action. Great organ (7.52%)
Simon Sinek: The Infinite Game: "In the Infinite Game, the true value of an organization (6.64%)
Jim Collins: Great by Choice: "20 Mile March. Achieve consistent performance markers, in (5.40%)
Jim Collins: Good to Great: "Implementing ETO and embracing BHAGs can significantly acce (5.17%)
Jim Collins: How the Mighty Fall: "Five stages of decline: hubris born of success, undis (4.33%)
Jim Collins: Built to Last: "Clock building, not time telling. Focus on building an orga (4.30%)
Jim Collins: Beyond Entrepreneurship 2.0: "The flywheel effect. Success comes from consi (3.60%)
Jim Collins: Built to Last: "Preserve the core, stimulate progress. Enduring great compa (3.40%)
Jim Collins: Good to Great: "Good is the enemy of great. To go from good to great requir (2.38%)
Algolia results:
Simon Sinek: Start with Why: "People don't buy what you do; the
Jim Collins: Turning the Flywheel: "Disciplined people, thought
Jim Collins: Good to Great: "First who, then what. Get the righ
Query: Find me everything about people
Author(s): Jim Collins, Simon Sinek

```

Nie można jednoznacznie powiedzieć, że wyniki Algolia są lepsze, ponieważ Qdrant na trzeciej i czwartej pozycji zwrócił książkę na temat przywództwa, co jest bezpośrednio powiązane z naszym zapytaniem.

Jeśli więc zastosujemy RRF do uporządkowania wyników, otrzymamy TOP3 z książkami zawierającymi frazę **people**, a wspomniane tytuły mówiące o przywództwie wylądują zaraz obok.

	Author	Text	RRF Score
0	Simon Sinek	Start with Why: "People don't buy what you do; they buy why you do it. And ...	1.5000
1	Jim Collins	Good to Great: "First who, then what. Get the right people on the bus, the ...	1.3333
2	Jim Collins	Turning the Flywheel: "Disciplined people, thought, and action. Great organ...	0.7000
3	Simon Sinek	Leaders Eat Last: "The true price of Leadership is the willingness to place...	0.3333
4	Simon Sinek	Leaders Eat Last: "Understanding the importance of OKRs and fostering psych...	0.2500
5	Simon Sinek	The Infinite Game: "In the Infinite Game, the true value of an organization...	0.1667
6	Jim Collins	Great by Choice: "20 Mile March. Achieve consistent performance markers, in...	0.1429
7	Jim Collins	Good to Great: "Implementing ETO and embracing BHAGs can significantly acce...	0.1250
8	Jim Collins	How the Mighty Fall: "Five stages of decline: hubris born of success, undis...	0.1111
9	Jim Collins	Built to Last: "Clock building, not time telling. Focus on building an orga...	0.1000
10	Jim Collins	Beyond Entrepreneurship 2.0: "The flywheel effect. Success comes from consi...	0.0909
11	Jim Collins	Built to Last: "Preserve the core, stimulate progress. Enduring great compa...	0.0833
12	Jim Collins	Good to Great: "Good is the enemy of great. To go from good to great requir...	0.0769

Bez wątpienia jest to najlepszy wynik, jaki mogliśmy uzyskać i był on możliwy tylko ze względu na zastosowanie wyszukiwania hybrydowego.

Są również sytuacje, gdzie skuteczność wyszukiwania nie będzie porównywalna dla obu strategii. Na przykład, jeśli zapytanie koncentruje się na akronimach, numerach seryjnych czy literówkach, to wyszukiwanie pełnotekstowe ma znacznie większą wartość.

Poniżej mamy przykład przeszukiwania za pomocą numeru ISBN (wygenerowałem losowe wartości). W tym przypadku baza wektorowa zwróciła dokument, ale dopiero na 8 z 9 pozycji. Z kolei Algolia zwróciła tylko jeden dokument. W rezultacie poszukiwana książka trafiła na 1 pozycję.

Fetching all documents
Found 13 documents
Vector results:

```

Jim Collins: Good to Great: "Good is the enemy of great. To go from good to great requir (40.72%)
Jim Collins: How the Mighty Fall: "Five stages of decline: hubris born of success, undis (40.49%)
Jim Collins: Great by Choice: "20 Mile March. Achieve consistent performance markers, in (40.18%)
Jim Collins: Good to Great: "Implementing ETO and embracing BHAGs can significantly acce (39.96%)
Jim Collins: Good to Great: "First who, then what. Get the right people on the bus, the (39.81%)
Jim Collins: Built to Last: "Clock building, not time telling. Focus on building an orga (38.50%)
Jim Collins: Built to Last: "Preserve the core, stimulate progress. Enduring great compa (38.11%)
Jim Collins: Turning the Flywheel: "Disciplined people, thought, and action. Great organ (36.87%) ←
Jim Collins: Beyond Entrepreneurship 2.0: "The flywheel effect. Success comes from consi (31.56%)
Algolia results:
Jim Collins: Turning the Flywheel: "Disciplined people, thought
Query: 978-0-06-29360-4 ←
Author(s): Jim Collins

```

	Author	Text	RRF Score
0	Jim Collins	Turning the Flywheel: "Disciplined people, thought, and action. Great organ...	1.1250 ←
1	Jim Collins	Good to Great: "Good is the enemy of great. To go from good to great requir...	1.0000
2	Jim Collins	How the Mighty Fall: "Five stages of decline: hubris born of success, undis...	0.5000
3	Jim Collins	Great by Choice: "20 Mile March. Achieve consistent performance markers, in...	0.3333
4	Jim Collins	Good to Great: "Implementing ETO and embracing BHAGs can significantly acce...	0.2500
5	Jim Collins	Good to Great: "First who, then what. Get the right people on the bus, the ...	0.2000
6	Jim Collins	Built to Last: "Clock building, not time telling. Focus on building an orga...	0.1667
7	Jim Collins	Built to Last: "Preserve the core, stimulate progress. Enduring great compa...	0.1429
8	Jim Collins	Beyond Entrepreneurship 2.0: "The flywheel effect. Success comes from consi...	0.1111

No i oczywiście będziemy mieć też odwrotne sytuacje, w których dopasowanie słów kluczowych w ogóle nie będzie występować. Wówczas Algolia nie zwróci nam żadnych rezultatów, a Qdrant z dużym prawdopodobieństwem odnajdzie właściwe dokumenty.

Wzbogacanie zapytań i self-query

Co jeśli jednak problem z dotarciem do pożądanych treści nie będzie leżał po stronie silnika wyszukiwania, lecz pojawi się już na etapie samego zapytania? W takiej sytuacji możemy zaangażować LLM, aby zmodyfikował oryginalne zapytanie, lub aby na jego podstawie wygenerował serię nowych zapytań.

Przykładem może być tak prosta interakcja, jak zadanie Agentowi AI pytania "Cześć, co tam?". Jest to ogólne zapytanie w przypadku którego trudno zarówno z pomocą wyszukiwarki odnaleźć potencjalnie istotne dokumenty.

Jeśli jednak model w prompcie weźmie pod uwagę np. porę dnia, bieżącą lokalizację czy bardzo ogólne informacje na swój temat, to możliwe jest wygenerowanie kilku pytań kierowanych 'do samego siebie'. W ten sposób otrzymujemy treść, która może już być wykorzystana do przeszukania bazy danych.



The screenshot shows a dark-themed conversational interface. The user's message 'Hey there! What's up?' is in a light blue box at the top. Below it, the AI's response is shown in a dark box:

```
{~  
read: [~  
  0: "What's my current status and mood as Alice?"  
  1: "Any recent interactions or important events with Adam?"  
  2: "What's the current context or situation for Adam based on the environment?"  
  3: "Do I have any pending updates or reminders for Adam?"  
]  
write: [~  
]  
}
```

Patrząc na powyższe zapytania, widzimy, że są one **spersonalizowane** i wynika to zastosowania początkowego kontekstu, który w tym przypadku jest opisem osobowości i najważniejszych informacji agenta. Uwzględnione są tam również ogólne zasady, takie jak:

- Zadaj pytania na temat wspomnianych osób, projektów czy narzędzi
- Gdy odpowiedź może dotyczyć Ciebie, wczytaj swój profil, wspominając swoje imię "Alice"
- Gdy pytanie dotyczy użytkownika, zawsze wymień jego imię (Adam)
- W przypadku ogólnych wiadomości, przeskanuj listę dostępnych kategorii w poszukiwaniu potencjalnego tematu, który może pomóc Ci kontynuować rozmowę i uczynić ją angażującą.

Podobne zasady znacznie wzbogacają interakcję z agentem zdolnym do adaptacji do różnych sytuacji. Warto też wspomnieć, że użycie słów kluczowych, takich jak imiona, jest celowe, ponieważ generowana lista ma służyć silnikom wyszukiwania.

Filtrowanie i przetwarzanie wyników wyszukiwania

W poprzedniej lekcji widzieliśmy już przykład re-rank realizowany przez model oceniający poszczególne wyniki pod kątem przydatności dla bieżącej interakcji. Oczywiście nic nie stoi na przeszkodzie, aby nadal korzystać z tej metody w połączeniu z wyszukiwaniem hybrydowym.

Natomiast mogą zdarzyć się także sytuacje w których nie będzie interesować nas precyzyjne wyszukanie kilku/kilkunastu dokumentów, ale pobranie wszystkich z danej kategorii. Wówczas do takiego zadania będziemy chcieli zaangażować albo wyłącznie bazę danych, albo silnik wyszukiwania taki jak algolia. Wówczas rolą LLM będzie zwrócenie np. identyfikatorów bądź nazw kategorii i filtrów, które mają być zastosowane.

Sytuacja, w której może być to potrzebne, dotyczy zapytań takich jak: "wypisz wszystkie książki autorów urodzonych w XXXX roku" lub "podsumuj wyniki raportów z lat 2010-2024". Wówczas wyszukiwarka zwraca dużą liczbę dokumentów, które zawierają także mnóstwo treści nieistotnej z punktu widzenia aktualnego zadania. Zamiast więc wczytywać je do kontekstu konwersacji, możemy skorzystać z logiki tworzącej podsumowanie skupiające się tylko na jednym zagadnieniu. Dodatkowo wynik takiego podsumowania, może zostać zachowany na potrzeby przyszłych interakcji, aby powtarzanie tego procesu nie było konieczne.

Zatem:

- LLM może decydować o tym, które obszary bazy danych **przefiltrować** w celu pobrania zestawu dokumentów pasujących do filtra, a nie konkretnego zapytania
- Listę pobranych dokumentów możemy przekazać do kontekstu bezpośrednio, lub przeprocesować w celu precyzyjnej ekstrakcji tylko wybranych danych
- Rezultat z przeprocesowanych dokumentów możemy zapisać w bazie jako **nowy dokument** w celu ograniczenia konieczności wykonywania tej samej akcji wielokrotnie

Implementacja powyższych punktów może obyć się na podstawie przykładów, które już zrealizowaliśmy (np. `summary`).

Dostarczanie wyników wyszukiwania do kontekstu

Metadane są przydatne nie tylko w celu filtrów na etapie wyszukiwania, ale przede wszystkim na etapie dostarczania treści dokumentu do kontekstu. Jego główna treść zwykle będzie niewystarczająca, aby LLM był w stanie skutecznie się nią posługiwać.

Poniższy prompt pokazuje w praktyce, jak ważna jest taka forma dostarczania dokumentów. Gdybyśmy nie uwzględnili daty utworzenia "created_at", model odpowiedziałby na pytanie użytkownika pozytywnie, a tak odmówił zgodnie z prawdą.

```
AI_devs 3

Answer questions based solely on the context below. If you don't know the answer,
say "I don't know."

<context>

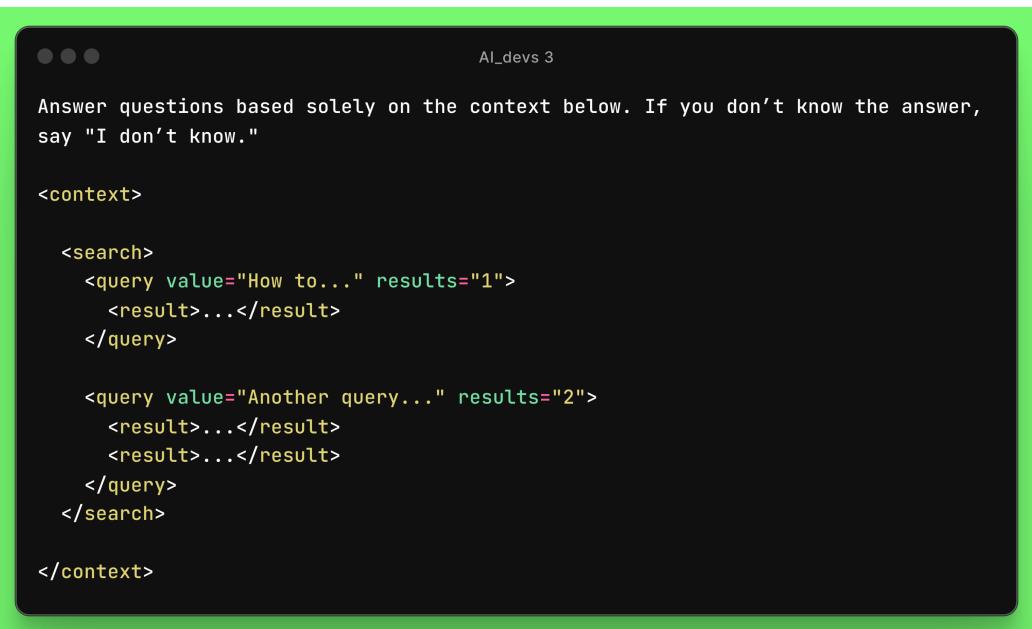
<doc
  created_at="2024-05-10"
  title="How to..."
  source="https://brain.overment.com"
>
...
</doc>

</context>

User: Do you have any docs from oct 2024?
AI: Nope, sorry
```

Podobnie, możemy oddzielać informacje pochodzące z różnych narzędzi, z których korzysta agent. Jednak nie warto tego nadużywać i zawsze powinniśmy dążyć do tego, aby w danym kontekście znajdowały się tylko najważniejsze dokumenty.

W formatowaniu kontekstu w taki sposób, przydatne okazuje się stosowanie składni zbliżonej do `xml`, ponieważ wyraźnie oddzielamy od siebie nie tylko poszczególne konteksty, ale także fragmenty treści znajdujące się wewnątrz nich. Przykładem może być lista wyników wyszukiwania, zawierająca rezultaty dla więcej niż jednego zapytania.



```
AI_devs 3

Answer questions based solely on the context below. If you don't know the answer,
say "I don't know."

<context>

<search>
  <query value="How to..." results="1">
    <result>...</result>
  </query>

  <query value="Another query..." results="2">
    <result>...</result>
    <result>...</result>
  </query>
</search>

</context>
```

Podsumowanie

Budowanie Retrieval-Augmented Generation bez zastosowania wyszukiwania hybrydowego od tej chwili praktycznie nie powinno być brane pod uwagę. Tym bardziej że skonfigurowanie wyszukiwania pełnotekstowego i późniejsza ocena wyników są dość proste.

Jednocześnie jasne powinno być to, że nawet najlepsze systemy wyszukiwania nie wystarczą, aby zaadresować możliwie dużą część zapytań i ich kategorii. To wszystko prowadzi więc do wniosku, że **uniwersalne systemy RAG** próbujące dopasować się do dowolnego rodzaju danych, są jeszcze poza naszym zasięgiem o ile zależy nam na bardzo wysokiej skuteczności.

Ogromną rolę pełnią także elementy wspierające proces wyszukiwania, takie jak wzbogacanie zapytania (np. [Self-Querying](#)), czy ocenianie rezultatów przez model.

Jeśli z tej lekcji masz zabrać ze sobą tylko jedną rzecz, to zapoznaj się bliżej z przykładem [hybrid](#) i spróbuj uruchomić go na własnych, bezpłatnych kontach [Algolia](#) i [Qdrant](#) w połączeniu z zestawem danych wygenerowanym przez Ciebie (możesz użyć do tego [LLM](#)).

Powodzenia!