Wektory jako alfabet sztucznej inteligencji: od baz danych do systemów RAG w ujęciu Nitina Borwankara

AUTOR

Fundacja Dobre Państwo

STRESZCZENIE / ABSTRACT

Artykuł analizuje ewolucję przechowywania danych: od tradycyjnych baz relacyjnych, pełniących funkcję cyfrowych archiwów, do nowoczesnych reprezentacji wektorowych. Autor wyjaśnia koncepcję embeddings jako procesu przekładania ludzkiego znaczenia na geometrię wielowymiarową, co pozwala maszynom na rozumienie semantyczne zamiast prostego dopasowywania słów kluczowych. W tekście podkreślono różnicę między wektorami rzadkimi a gęstymi oraz wskazano, że wektory nie „rozumieją" w ludzkim sensie, lecz kodują matematyczne regularności i podobieństwa. To przejście od indeksowania do mapowania kontekstu stanowi fundament systemów RAG i nowoczesnej infrastruktury AI, umożliwiając maszynom operowanie na pokrewieństwie znaczeń, a nie tylko na identyczności znaków.

The article analyzes the evolution of data storage: from traditional relational databases, serving as digital archives, to modern vector representations. The author explains the concept of embeddings as a process of translating human meaning into multi-dimensional geometry, allowing machines to achieve semantic understanding instead of simple keyword matching. The text emphasizes the difference between sparse and dense vectors and points out that vectors do not 'understand' in the human sense, but rather encode mathematical regularities and similarities. This transition from indexing to context mapping forms the foundation of RAG systems and modern AI infrastructure, enabling machines to operate on kinship of meaning rather than just character identity.

Artykuł analizuje fundamentalną transformację w architekturze systemów AI: przejście od tradycyjnego, statycznego archiwizowania danych ku dynamicznemu rozumieniu

semantycznemu. Autor stawia tezę, że współczesna sztuczna inteligencja przestaje być jedynie elokwentnym improwizatorem, a staje się odpowiedzialnym narzędziem analitycznym dzięki synergii wektorów, baz wektorowych i architektury RAG (Retrieval-Augmented Generation). Kluczowym elementem tej ewolucji jest zastąpienie sztywnej identyczności słów kluczowych geometrią znaczeń, gdzie wektory pozwalają maszynom operować na podobieństwach i kontekstach. Poprzez analizę konkretnych implementacji – od lokalnych systemów opartych na SQLite po zaawansowane wyszukiwarki naukowe w PostgreSQL – tekst dowodzi, że uziemienie (grounding) modeli językowych w konkretnych źródłach wiedzy jest jedynym skutecznym sposobem na eliminację halucynacji i zapewnienie wiarygodności odpowiedzi. W ten sposób infrastruktura pamięci staje się nową gramatyką danych, która nie zastępuje rygoru baz relacyjnych, lecz dopełnia go, tworząc hybrydowe systemy zdolne do zarządzania wiedzą w sposób autonomiczny i bezpieczny.

SŁOWA KLUCZOWE

wektory | systemy RAG | embeddings | bazy wektorowe | podobieństwo kosinusowe | FAISS | Approximate Nearest Neighbor | HNSW | kwantyzacja produktowa | VQL | lokalne LLM | rozumienie semantyczne | halucynacje AI | architektura pamięci organizacyjnej | multihop reasoning

Wektory jako przejście od archiwizacji do rozumienia semantycznego

Od danych martwych do danych znaczących. Dlaczego wektor stał się alfabetem sztucznej inteligencji

Przez większą część historii informatyki dane traktowano jak materiał administracyjny: należało je zapisać, uporządkować, odnaleźć, policzyć, przenieść, zabezpieczyć i odtworzyć. Baza danych przez dekady nie była więc tyle mózgiem systemu, co jego archiwum. Przechowywała rekordy, tabele, pola, klucze, indeksy i relacje. Stanowiła wielką kartotekę nowoczesności, bibliotekę rubryk, urzędowym snem o świecie, w którym wszystko ma swój numer, typ, datę, właściciela i status. Ten

model doskonale obsługiwał dane ustrukturyzowane: fakturę, PESEL, saldo, nazwisko, adres, cenę, kategorię produktu czy datę operacji.

Gorzej radził sobie z tym, co w ludzkim świecie najważniejsze: z sensem. Człowiek rzadko pyta bowiem w sposób idealnie zgodny z rubryką. Nie szuka wyłącznie dokumentu, w którym występuje słowo X, lecz pyta: „który tekst mówi o tym samym problemie, choć używa innych pojęć?". Nie pyta tylko o frazę „uczenie maszynowe", ale szuka wyjaśnienia mechanizmu podobnego do tego, o którym myśli. Nie poszukuje identyczności liter, lecz pokrewieństwa znaczeń. W tym punkcie klasyczna informatyka wyglądała jak urzędnik, który słyszy muzykę, lecz sprawdza tylko numery taktów. Formalnie poprawnie, poznawczo ubogo.

Wektor w nowoczesnej sztucznej inteligencji jest odpowiedzią na ten deficyt. W najprostszym ujęciu to uporządkowana lista liczb – zazwyczaj zmiennoprzecinkowych – które reprezentują obiekt w wielowymiarowej przestrzeni. Obiektem może być słowo, zdanie, akapit, obraz, nagranie, fragment kodu, profil użytkownika, publikacja naukowa lub cały dokument. W sensie głębszym wektor jest próbą przełożenia jakości na ilość: zamianą ludzkiego znaczenia w geometrię, którą maszyna może porównywać, mierzyć i przeszukiwać.

To nie jest już martwy BLOB wrzucony do bazy jak zamknięta paczka, lecz obiekt posiadający położenie w przestrzeni sensów. W klasycznym wyszukiwaniu słów kluczowych dokumenty są bliskie wtedy, gdy zawierają podobne frazy. W wyszukiwaniu semantycznym mogą być blisko nawet wtedy, gdy mówią innymi słowami o tym samym. Zdanie „film był fantastyczny" i zdanie „kino było świetne" nie muszą dzielić niemal żadnych istotnych tokenów, a jednak człowiek natychmiast rozpoznaje ich pokrewieństwo.

Dla systemu opartego na embeddings mogą one również znaleźć się blisko siebie w przestrzeni wektorowej. To moment, w którym baza danych przestaje być wyłącznie katalogiem, a zaczyna działać jak urządzenie do przybliżonego rozumienia. Słowo „przybliżonego" jest tu kluczowe. Wektor nie jest duszą pojęcia, platońską ideą ani gwarancją prawdy. Jest modelem, a model zawsze stanowi negocjację między rzeczywistością a kosztem jej obliczeniowego przedstawienia. Wektory nie rozumieją w ludzki sposób; one kodują regularności wykryte w danych. Jeśli dane są bogate i różnorodne, przestrzeń wektorowa oddaje subtelne relacje znaczeniowe. Jeśli są ubogie lub stronnicze, wektor przejmie tę ślepotę i będzie ją wykonywał z beznamiętną elegancją matematyki. Maszyna nie posiada sumienia, lecz funkcję podobieństwa – to zasadnicza różnica, której nie wolno przykrywać marketingowym brokatem.

Mechanizm embeddings, czyli osadzania, polega na mapowaniu obiektów do przestrzeni wektorowej. Jako czasownik „embedding" oznacza proces przekształcania danych w wektory; jako

rzeczownik – wynik tego procesu: gotową reprezentację liczbową. Ta podwójność pokazuje, że nie mówimy jedynie o typie danych, lecz o całej epistemologii technicznej: o tym, jak system AI ustanawia swoje „widzenie" świata.

Embedding jest soczewką. Może wyostrzać sens, ale może też deformować. Pomaga odkrywać podobieństwa, lecz może ukrywać różnice decydujące dla prawnika, lekarza czy historyka. Historia tego przejścia zaczyna się od rozczarowania wektorami rzadkimi. Dawne reprezentacje tekstu, takie jak TF-IDF czy „worki słów", były użyteczne, lecz semantycznie prymitywne. Tworzyły ogromne przestrzenie, w których większość wymiarów była zerowa, a obecność słowa znaczyła więcej niż jego sens. Była to informatyka indeksu, nie znaczenia.

Wektory gęste odwróciły tę logikę. Zamiast setek tysięcy pustych wymiarów wykorzystują kilkaset lub kilka tysięcy wymiarów nasyconych informacją. Pojedyncza liczba nie ma prostego ludzkiego odpowiednika, ale cały wektor reprezentuje kontekst, intencję i relacje. Przełomowy Word2Vec udowodnił, że relacje semantyczne mogą mieć strukturę geometryczną. Słynne równanie z królem i królową stało się memem technologicznym, lecz pod spodem kryje się poważna intuicja: znaczenie nie musi być definicją słownikową – może być położeniem i kierunkiem w przestrzeni. To dlatego arytmetyka wektorowa wydaje się niemal magiczna. Jednak magia ta jest matematyką, której nie zdążyliśmy jeszcze społecznie oswoić.

Wektory jako infrastruktura podobieństwa i kontekstu

Późniejsze modele, takie jak Doc2Vec, a następnie architektury transformatorowe, przesunęły akcent z pojedynczego słowa na szersze jednostki znaczenia: zdania, akapity, dokumenty i sekwencje kontekstowe. W transformerach osadzanie przestało być zwykłym przypisaniem wektora do słowa; stało się syntezą kilku warstw informacji: znaczenia tokenu, jego pozycji w sekwencji oraz funkcji w danym segmencie wejścia. To właśnie dzięki temu współczesne modele językowe reagują na kontekst znacznie subtelniej niż starsze systemy. Słowo „zamek" w tekście o architekturze i „zamek" w opisie kurtki nie powinny prowadzić systemu w to samo miejsce semantyczne – i właśnie tę różnicę stara się uchwycić sprawny mechanizm osadzania.

Teza Borwankar, że wektory są „medium, w którym sztuczna inteligencja myśli", jest nośna, choć wymaga ostrożności. Jako metafora sprawdza się znakomicie, jednak jako twierdzenie ontologiczne byłaby przesadą. AI nie myśli w ludzkim sensie; wykonuje operacje na reprezentacjach, które pozwalają jej symulować rozumienie. Wektory są więc nie tyle myślą, co środowiskiem obliczeniowym, w którym podobieństwo zaczyna pełnić funkcję poznawczą. Przypomina to mapę miasta, w której ulice nie oznaczają asfaltu, lecz skojarzenia, a odległość między punktami

wskazuje, jak blisko siebie leżą dwa sensy. Mapa ta została jednak sporządzona przez kartografa, który nigdy nie był w terenie, lecz przeczytał miliardy opisów podróży.

Z tego wynika pierwsza kluczowa konsekwencja: baza wektorowa nie jest po prostu szybszą bazą danych. Jest instytucją pośredniczącą między pamięcią a interpretacją. Klasyczna baza odpowiadała na pytanie: „czy taki rekord istnieje?". Baza wektorowa pyta raczej: „co jest do tego podobne?". Przejście z identyczności na podobieństwo jest cywilizacyjnie niebanalne. Prawo, administracja i księgowość opierają się na identyczności, gdyż daje ona dowód. Kultura, nauka i język żyją natomiast podobieństwem, analogią, kontekstem i rodzinami znaczeń. Sztuczna inteligencja potrzebuje więc infrastruktury zdolnej obsłużyć nie tylko „to samo", ale także „prawie to samo", „w tym samym duchu" czy „semantycznie blisko". To właśnie powód, dla którego wektory stały się alfabetem współczesnych systemów AI.

Drugą konsekwencją jest konieczność wyboru metryki. Skoro sens został przedstawiony geometrycznie, trzeba zdecydować, jak mierzyć bliskość. Podobieństwo kosinusowe bada kąt między wektorami, ignorując ich długość; w analizie tekstu jest to często pożądane, by dłuższy dokument nie zyskał przewagi nad krótszym jedynie przez większą „masę" liczbową. Odległość euklidesowa mierzy dystans między punktami, co lepiej sprawdza się w zadaniach matematycznych i klastrowych. Iloczyn skalarny bywa z kolei użyteczny w systemach rekomendacyjnych, gdzie liczy się nie tylko kierunek podobieństwa, ale i intensywność preferencji. Metryka nie jest zatem neutralnym detalem technicznym. Jest decyzją epistemologiczną zapisaną w kodzie.

Trzecia konsekwencja dotyczy kosztów. Reprezentacje wektorowe są potężne, lecz kosztowne. Im więcej wymiarów, tym bogatsza przestrzeń znaczeniowa, ale i większe zużycie pamięci oraz dłuższy czas przeszukiwania, co wymusza stosowanie wyspecjalizowanych indeksów. O ile na małej próbce można porównywać każdy element z każdym, o tyle przy milionach czy miliardach obiektów metoda brutalnej siły staje się obliczeniowym barbarzyństwem.

Infrastruktura pamięci jako pas bezpieczeństwa dla LLM

Konieczne jest zatem wprowadzenie algorytmów przybliżonego najbliższego sąsiada (ANN). To tutaj otwiera się świat FAISS, HNSW, IVF, kwantyzacji oraz nieustannych kompromisów między szybkością a dokładnością. W tym punkcie staje się jasne, że nowoczesna sztuczna inteligencja nie jest wyłącznie triumfem modeli językowych, lecz także infrastruktury wyszukiwania. Wielki model pozbawiony dostępu do właściwej pamięci pozostaje elokwentnym improwizatorem. Potrafi mówić pięknie, ale gdy nie zna odpowiedzi, zaczyna konfabulować z miną profesora po trzeciej kawie.

Dopiero połączenie modelu językowego z przemyślaną bazą wiedzy tworzy system zdolny nie tylko do generowania tekstu, ale do odpowiadania w oparciu o konkretne dokumenty.

Właśnie dlatego RAG nie jest jedynie dodatkiem do LLM – to próba założenia modelowi językowemu pasa bezpieczeństwa. Borwankar w swojej koncepcji zmierza wyraźnie w stronę praktycznej, lokalnej i kontrolowalnej sztucznej inteligencji. Nie interesuje go abstrakcyjna fascynacja wektorami, lecz budowa działających systemów: osobistej wyszukiwarki wiedzy, naukowego asystenta literaturowego, lokalnego RAG-a czy hipotetycznego języka VQL, który mógłby uporządkować rozdrobniony rynek baz wektorowych. Oficjalny opis książki potwierdza to praktyczne podejście: od podstaw teorii embeddings i Word2Vec, przez SQLite i PostgreSQL, aż po aplikacje RAG.

Taka perspektywa jest kluczowa, gdyż w debacie o AI zbyt często grzęźnimy między dwiema karykaturami. Jedni widzą w sztucznej inteligencji niemal metafizyczną istotę, która lada moment posiądzie rozum absolutny. Drudzy redukują ją do „statystycznego papugowania", ignorując fakt, że skala i architektura operowania na reprezentacjach zmieniają jakość procesu. Bazy wektorowe pozwalają wyjść poza ten jałowy teatr. Pokazują, że najważniejsze procesy zachodzą nie na scenie, gdzie model wypowiada odpowiedź, lecz za kulisami, gdzie system selekcjonuje fragmenty świata, które zostaną podane jako kontekst. Jeśli chcemy poważnie rozumieć AI, musimy zejść z poziomu zachwytu nad interfejsem do poziomu infrastruktury pamięci.

Pytanie nie brzmi zatem tylko: „jaki model odpowiada?", lecz także: „z jakiej bazy korzysta?", „jak pocięto dokumenty?", „jakim modelem wykonano embeddings?", „jaką metryką mierzono podobieństwo?". Należy zapytać, czy zastosowano wyszukiwanie hybrydowe, czy wyniki były rerankowane, czy model musiał cytować źródła i czy pracował lokalnie, czy wysłał prywatną wiedzę do cudzej chmury. Dopiero te pytania oddzielają realną inżynierię wiedzy od cyfrowego spirytualizmu. Wektory stają się nową gramatyką danych. Nie zastępują klasycznych baz relacyjnych, lecz dopełniają je tam, gdzie świat nie mieści się w rubrykach. Relacyjne bazy danych pozostają niezastąpione w kwestiach transakcji, spójności i twardych filtrów; bazy wektorowe wnoszą natomiast możliwość pracy na znaczeniu, podobieństwie i kontekście.

Najciekawsze systemy przyszłości nie będą wybierać między tymi podejściami. Będą hybrydowe: połączą rygor SQL z elastycznością przestrzeni semantycznej, filtr metadanych z wyszukiwaniem sensu, dowód z analogią i porządek tabeli z dynamiką języka. W tym miejscu zaczyna się właściwa opowieść o FAISS oraz indeksach, które pozwalają przeszukiwać wielowymiarowe przestrzenie szybciej, niż pozwalałaby na to naiwna matematyka. Jeśli embedding jest aktem nadania znaczenia postaci liczbowej, to wyszukiwanie najbliższego sąsiada jest aktem odnajdywania pokrewieństwa.

W epoce, w której każdy dokument czy obraz może zostać przekształcony w wektor, pytanie o szybkość i trafność znajdowania podobieństw staje się pytaniem o nową władzę nad wiedzą.

FAISS i ANN jako rozwiązanie problemu skalowalności wyszukiwania

Jeśli operujemy na dziesięciu zdaniach zamienionych na wektory, możemy porównać zapytanie z każdym z nich i wybrać najbardziej podobne. Przy tysiącu dokumentów wciąż da się to zrobić bez większej tragedii. Jednak gdy w grę wchodzą miliony lub miliardy fragmentów tekstu, obrazów, profili czy publikacji naukowych, brutalne porównywanie wszystkiego ze wszystkim staje się obliczeniowym młóceniem oceanu widelcem. W teorii rozwiązanie to działa; w praktyce kompromituje projekt, budżet i cierpliwość użytkownika.

W tym miejscu pojawia się FAISS (Facebook AI Similarity Search) – biblioteka rozwijana przez środowisko Meta, zaprojektowana do wydajnego klastrowania i wyszukiwania podobieństw w gęstych wektorach. FAISS odpowiada na fundamentalne wyzwanie nowoczesnej AI: jak zarządzać ogromnymi zbiorami danych wektorowych, których tradycyjne podejście brute-force nie jest w stanie obsłużyć ani czasowo, ani pamięciowo. Nie jest to zatem zwykła biblioteka pomocnicza, lecz jeden z tych cichych silników, dzięki którym semantyczne wyszukiwanie przestaje być pokazem laboratoryjnym, a staje się usługą możliwą do wdrożenia w realnym systemie.

Kluczowym pojęciem jest tu ANN (Approximate Nearest Neighbor), czyli przybliżony najbliższy sąsiad. Brzmi to niepozornie, ale kryje w sobie całą filozofię współczesnej inżynierii AI: nie zawsze potrzebujemy wyniku absolutnie idealnego, jeśli możemy otrzymać wynik bardzo dobry w czasie tysiące razy krótszym. W tradycyjnej matematyce mogłoby to uchodzić za ustępstwo; w praktyce systemów AI jest to warunek przetrwania.

Algorytm przybliżony rezygnuje z gwarancji znalezienia jednego, obiektywnie najbliższego wektora w całej przestrzeni. W zamian oferuje sąsiadów wystarczająco bliskich i osiągalnych w czasie akceptowalnym dla użytkownika. To moment, w którym należy porzucić dziecinne marzenie o absolutnej precyzji. W systemach semantycznych „najlepszy” wynik rzadko jest jeden – język naturalny jest rozmyty, pytania bywają niedookreślone, a znaczenia mają rodzinne podobieństwa, nie wojskowe numery seryjne.

Jeżeli użytkownik pyta o „systemy ograniczające halucynacje LLM przez dostęp do dokumentów”, trafną odpowiedzią mogą być teksty operujące pojęciami RAG, retrieval, grounding, vector search czy semantic memory. Dokładność w takim świecie nie polega na mechanicznym odnalezieniu

jednego rekordu, lecz na stworzeniu trafnej puli kandydatów, z której system zbuduje sensowną odpowiedź. FAISS oferuje kilka metod organizacji tej puli.

Najprostsze są indeksy płaskie (Flat Indexes). Ich zaletą jest stuprocentowa dokładność, wynikająca z porównania zapytania ze wszystkimi wektorami. Ta sama cecha jest jednak ich największą wadą: dla dużych baz danych to eleganckie samobójstwo wydajności. Indeks płaski przypomina bibliotekarza, który na każde pytanie czytelnika otwiera każdą książkę w magazynie, by sprawdzić, czy nie ma w niej odpowiedzi – skrupulatność godna podziwu, organizacja pracy fatalna.

Dlatego w praktyce stosuje się struktury zawężające przestrzeń poszukiwania, takie jak IVF (Inverted File Index). Jego logika opiera się na podziale przestrzeni wektorowej na klastry, zazwyczaj przy użyciu algorytmu k-means. Zamiast przeszukiwać całość, system najpierw ustala, które klastry są najbliższe zapytaniu, a następnie szuka kandydatów tylko w ich obrębie. Parametr nprobe decyduje o liczbie sprawdzanych klastrów: im jest wyższy, tym większa szansa na trafienie idealnego wyniku, ale i wyższy koszt obliczeniowy. To znów potwierdza podstawowe prawo tej dziedziny: jakość, szybkość i pamięć tworzą trójkąt, w którym nie da się bezkarnie maksymalizować wszystkich parametrów jednocześnie.

Jeszcze bardziej sugestywnym rozwiązaniem jest HNSW (Hierarchical Navigable Small World). Można go opisać za pomocą metafory autostrad i dróg lokalnych: wysokie warstwy grafu pozwalają na duże skoki w kierunku obszaru, w którym prawdopodobnie znajduje się odpowiedź, natomiast niższe warstwy umożliwiają precyzyjne dotarcie do najbliższych sąsiadów. To nie jest przeszukiwanie kartoteki od A do Z, lecz nawigacja po mieście – system najpierw wybiera autostradę, potem zjazd i ulicę, a na końcu konkretny numer domu. HNSW stanowi jeden z najważniejszych algorytmów praktycznego wyszukiwania wektorowego, oferując optymalny kompromis między szybkością a trafnością.

Systemy semantyczne jako maszyna inżynieryjnych kompromisów

Ten kompromis wymaga jednak precyzyjnego dostrojenia. Parametr M określa liczbę połączeń przypadających na węzeł grafu – ich zwiększenie może podnieść jakość wyszukiwania, ale wiąże się z większym zużyciem pamięci. Z kolei efConstruction wpływa na proces budowy indeksu: im wyższa wartość, tym lepszy graf, lecz wolniejszy czas indeksowania. Parametr efSearch determinuje natomiast jakość samego zapytania – wyższe ustawienie zwiększa dokładność kosztem szybkości odpowiedzi.

Nie istnieje zatem uniwersalne ustawienie; pozostaje jedynie inżynierska dyscyplina testowania. Jest to kluczowe, gdyż w dyskusjach o AI często mówi się o modelach tak, jakby problem sprowadzał się wyłącznie do wyboru „najmądrzejszego mózgu". Tymczasem systemy semantyczne wymagają również dobrze zaprojektowanego układu krążenia. Źle dobrany indeks sprawi, że nawet wysokiej jakości embeddings będą zwracać wyniki zbyt wolno, zbyt kosztownie lub zbyt płytko. Dobry indeks przypomina logistykę państwa: nikt nie skanduje jego nazwy na placach, ale gdy zawodzi, cała reszta zaczyna wyglądać jak teatr pozorów.

Drugim krytycznym wyzwaniem jest pamięć. Gęste wektory są kompaktowe w porównaniu do dawnych reprezentacji rzadkich, jednak przy ogromnej skali wciąż pochłaniają potężne zasoby. Jeśli każdy dokument lub chunk tekstu opisujemy kilkuset lub kilku tysiącami liczb zmiennoprzecinkowych, a takich fragmentów są miliony, rachunek pamięciowy przestaje być kwestią akademicką. W tym miejscu pojawia się kwantyzacja – kontrolowane obniżenie precyzji reprezentacji liczbowej w celu drastycznego zmniejszenia zużycia zasobów. Kwantyzacja skalarna (SQ) kompresuje każdy wymiar wektora osobno, redukując na przykład 32-bitowe floaty do 8-bitowych wartości całkowitych.

Jest to operacja stratna, lecz często akceptowalna. W wyszukiwaniu semantycznym nie zawsze wymagana jest mikroskopijna dokładność położenia punktu; czasem wystarczy wiedzieć, w którym rejonie się on znajduje. Kwantyzacja produktowa (PQ) idzie o krok dalej: dzieli wektor na mniejsze subwektory i koduje je za pomocą osobnych książek kodowych. Pozwala to osiągnąć bardzo wysoką kompresję i przyspieszyć obliczanie przybliżonych odległości. Cena jest oczywista: część dokładności zostaje poświęcona na ołtarzu skali.

To ponowna lekcja zdrowego sceptycyzmu. System AI nie jest bezcielesnym rozumem szybującym nad światem, lecz maszyną kompromisów. Każda decyzja projektowa generuje koszt: model embeddings, wymiarowość wektora, metryka odległości, rodzaj indeksu, kwantyzacja, chunking, filtr metadanych, próg podobieństwa, reranking czy temperatura LLM. Gdy ktoś obiecuje „inteligentną wyszukiwarkę", warto zapytać: inteligentną według jakiej metryki, jakiego indeksu, jakiego progu i jakiego modelu osadzania?

Dobór metryki i optymalizacja to rzemiosło wydajności

Diabeł nie siedzi już w szczegółach. Diabeł zrobił tam doktorat, prowadzi repozytorium i ma własny benchmark. W źródłach pojawia się istotna kwestia normalizacji wektorów: przy podobieństwie kosinusowym liczy się kierunek, a nie długość. Jeśli wektory zostaną znormalizowane do jedności, obliczanie podobieństwa można sprowadzić do szybkiego iloczynu skalarnego.

To przykład zasady, która w rzetelnych systemach technicznych powraca nieustannie: matematyczna elegancja przekłada się na wydajność operacyjną. Nie chodzi o ozdobne wzory, lecz o to, że poprawne rozumienie geometrii danych pozwala oszczędzać czas, pamięć i pieniądze. Metryki odległości nie są jednak tylko narzędziami obliczeniowymi – są sposobami definiowania podobieństwa. Dla tekstu najczęściej wybiera się podobieństwo kosinusowe, gdyż pozwala ono ignorować długość dokumentu i skupić się na kierunku semantycznym. W systemach rekomendacyjnych często lepszy okazuje się iloczyn skalarny, ponieważ uwzględnia on nie tylko podobieństwo, ale i siłę preferencji.

Dla zadań czysto geometrycznych, jak niektóre odmiany klastrowania, sensowna może być odległość euklidesowa. Innymi słowy: nie istnieje jedna, naturalna miara podobieństwa; istnieje jedynie miara adekwatna do konkretnego zadania. Kto nie rozumie zadania, ten wybiera metrykę jak kolor tapety: z gustu, nie z rozumu.

W tym miejscu widać już, dlaczego Borwankar tak mocno akcentuje praktyczne heurystyki. Jego podejście nie polega na budowaniu kolejnej mgławicy pojęć wokół AI, lecz na pokazywaniu, jak tworzyć systemy, które faktycznie działają: przetwarzaj dane wsadowo, zarządzaj długością dokumentów, normalizuj wektory, używaj GPU tam, gdzie ma to sens, pilnuj pamięci, dobieraj metrykę do domeny, testuj parametry HNSW, nie tnij tekstu bezmyślnie i wymuszaj cytowania w RAG-u. To nie są drobne rady techniczne. To etyka dobrego rzemiosła w epoce, w której źle zaprojektowany rurociąg danych może produkować pięknie brzmiące bzdury z prędkością przemysłową.

Szczególnie ważna jest zasada przetwarzania wsadowego. Przy masowym generowaniu embeddingów nie należy wektoryzować tekstów pojedynczo, gdyż takie podejście marnuje zasoby i spowalnia system. Batch processing pozwala wykorzystać architekturę obliczeniową znacznie efektywniej. W praktyce różnica jest kolosalna: oddziela systemy eksperymentalne od tych nadających się do realnej pracy. W informatyce, podobnie jak w instytucjach publicznych, nie wystarczy mieć słuszny cel; trzeba jeszcze nie zabić go procedurą. Drugą zasadą jest mądre zarządzanie długością tekstu – modele embeddingowe mają bowiem swoje limity wejścia.

Architektura indeksu i zarządzanie progiem podobieństwa jako fundamenty wiarygodności

Jeżeli dokument jest zbyt długi, może zostać po cichu ucięty. To zdradliwe rozwiązanie, ponieważ system nadal funkcjonuje, lecz działa na amputowanym sensie. Użytkownik otrzymuje wyniki, nie

wiedząc, że końcówka treści nigdy nie została prawidłowo uwzględniona. Dlatego chunking nie jest kosmetyką – jest warunkiem integralności poznawczej. Źle pocięty dokument przypomina zeznanie świadka, któremu regularnie wycinano ostatnie zdania akapitów. Niby mówi, ale sens zaczyna kuleć.

Trzecia zasada dotyczy GPU i pamięci. Karty graficzne radykalnie przyspieszają tworzenie embeddingów, jednak ich pamięć jest ograniczona i kosztowna. Przy dużych zbiorach niezbędna jest dbałość o czyszczenie pamięci podręcznej i kontrolę batchy. To techniczny detal tylko dla kogoś, kto nigdy nie uruchamiał realnego systemu. W praktyce wiele projektów AI przegrywa nie na poziomie idei, lecz zarządzania zasobami. Filozofia była piękna, pipeline zdechł. Epitafium startupu w siedmiu słowach.

FAISS pokazuje, że nowoczesne wyszukiwanie semantyczne to nie tylko kwestia „inteligencji", ale przede wszystkim architektury indeksu. Bez niego przestrzeń sensów staje się zbyt obszerna, by można ją było skutecznie przemierzać. Dzięki indeksom budujemy systemy, które znajdują podobne dokumenty, rekomendują treści, wykrywają duplikaty znaczeniowe, grupują publikacje, obsługują RAG czy wspierają wyszukiwanie wielomodalne i identyfikację zbliżonych obrazów. W każdym z tych przypadków mechanizm jest ten sam: nie szukamy już identycznego znaku, lecz zbliżonego znaczenia.

Ta zmiana uderza w klasyczne rozumienie baz danych. Tradycyjna baza była miejscem jednoznacznego przechowywania informacji. Baza wektorowa wprowadza element prawdopodobieństwa, podobieństwa i rankingu. Wynik nie jest po prostu „znaleziony" lub „nieznaleziony". Posiada on odległość, score, próg i pozycję w rankingu. Jest bardziej kandydatem niż werdyktem.

Wymaga to innej kultury korzystania z systemów informatycznych. Użytkownik musi rozumieć, że semantyczna wyszukiwarka nie jest wyrocznią, lecz maszyną do proponowania prawdopodobnie trafnych sąsiedztw znaczeniowych. Właśnie dlatego w systemach RAG próg podobieństwa ma kluczowe znaczenie. Jeśli będzie zbyt niski, system wstrzyknie do modelu LLM fragmenty luźno powiązane z pytaniem, co może prowadzić do odpowiedzi pozornie uzasadnionych, lecz opartych na słabych podstawach. Z kolei zbyt wysoki próg sprawi, że system nie odnajdzie wystarczającego kontekstu i odpowie zbyt ubogo. Dobry projektant musi zatem dobrać próg do domeny, rodzaju danych i poziomu ryzyka. W medycynie, prawie czy finansach tolerancja dla fałszywej pewności powinna być znacznie niższa niż w przypadku rekomendacji memów. Choć, uczciwie mówiąc, szkody społeczne po złych memach też bywają niedoszacowane.

Konieczność hybrydowości i przejście ku architekturze systemowej

FAISS i podobne narzędzia nie rozwiązują jednak całego problemu. Dostarczają sprawnego mechanizmu wyszukiwania wektorowego, ale realne systemy wymagają także metadanych, filtrów, autoryzacji, wersjonowania, aktualizacji, usuwania rekordów, audytu oraz integracji z klasycznymi bazami. Sama przestrzeń wektorowa wie, co jest podobne, lecz nie zawsze rozstrzyga, co jest dozwolone, aktualne, prawnie dostępne, przeznaczone dla konkretnego użytkownika czy zgodne z polityką organizacji. Dlatego praktyczne systemy nie mogą być czysto wektorowe – muszą być hybrydowe.

Tu zaczyna się przejście do dwóch projektów, które Borwankar przedstawia jako laboratoria tej filozofii: systemu wiedzy opartego na Reddit i SQLite-vss oraz naukowej wyszukiwarki ArXiv zbudowanej na PostgreSQL i pgvector. Pierwszy z nich obrazuje, jak stworzyć osobistą, lokalną pamięć semantyczną z chaotycznych danych społecznościowych. Drugi pokazuje budowę akademickiej wyszukiwarki sensu, zdolnej do pracy z abstraktami, sekcjami i fragmentami publikacji naukowych. Oba przykłady są istotne, gdyż ukazują dwie strony tej samej rewolucji: prywatną i instytucjonalną.

W systemie Reddit kluczowe znaczenie mają lekkość i lokalność. SQLite jest bazą prostą i przenośną, działającą w jednym pliku, a dzięki sqlite-vss zyskuje „wektorowe supermoce". To rozwiązanie nie udaje chmurowego molocha; jest raczej cyfrowym notesem na sterydach: można w nim przechowywać osobiste archiwa, zapisane posty, notatki i rozmowy, a następnie przeszukiwać je semantycznie. Dla osoby traktującej wiedzę jako zasób prywatny, a nie darmową rudę do wydobycia przez cudzy model biznesowy, jest to kierunek szczególnie ważny.

W systemie ArXiv akcent przesuwa się ku rygorowi naukowemu. Publikacje nie są chaotycznymi postami z forum – posiadają tytuły, autorów, abstrakty, metody i tabele; wymagają struktury. PostgreSQL z pgvector pozwala połączyć stabilność relacyjnej bazy danych z wyszukiwaniem podobieństwa. Można filtrować dane po autorze czy dacie, a jednocześnie szukać semantycznie w treściach prac. To właśnie przyszłość profesjonalnego wyszukiwania: nie wyrzucenie SQL-a na śmietnik, lecz jego rozszerzenie o zdolność pracy z sensem.

FAISS jest zatem jednym z fundamentów tej przyszłości, ale nie jej całością. Jest narzędziem do szybkiego poruszania się po przestrzeni wektorowej, jednak samo w sobie nie rozstrzyga, jakie dane warto wektoryzować, jak je ciąć, czyścić, filtrować czy chronić prywatność. Tu zaczyna się architektura systemowa.

Dlatego kolejny etap opowieści musi przejść od algorytmów do rurociągów: od tego, jak znaleźć podobieństwo, do tego, jak zbudować z niego prywatną pamięć i naukową wyszukiwarkę oraz system RAG, który nie zachowuje się jak gawędziarz po dopingu, lecz jak asystent trzymany za kołnierz przez źródła.

Przyglądając się bliżej SQLite-vss i osobistemu systemowi wiedzy, widać, że po omówieniu wektorów i FAISS trzeba zejść z poziomu ogólnej infrastruktury do konkretnego rurociągu. Dopiero wtedy okazuje się, że baza wektorowa nie jest magicznym pudełkiem, które po wrzuceniu danych „rozumie świat". Jest raczej dobrze zaprojektowanym warsztatem: każdy etap ma znaczenie, a pozorna drobnostka potrafi rozbić cały mechanizm.

Borwankar ilustruje to na przykładzie systemu zarządzania wiedzą opartego na treściach z Reddita i bazie SQLite. W notatce źródłowej projekt ten opisano jako lokalną wyszukiwarkę semantyczną, która rozumie znaczenie słów, a nie tylko dopasowuje litery.

Wybór Reddita jako materiału testowego jest znaczący. To przestrzeń półchaotyczna, językowo nierówna, pełna ironii i odpowiedzi wyrwanych z kontekstu – w skrócie: Reddit przypomina prawdziwy internet. A prawdziwy internet nie zachowuje się jak podręcznik do baz danych; jest raczej targowiskiem sensów, gdzie obok głębokiej obserwacji leży cyfrowa cebula. Dlatego projekt ten ma znaczenie szersze niż techniczna demonstracja. Pokazuje, jak AI może pracować z wiedzą nieidealną i fragmentaryczną. Współczesny człowiek przechowuje wiedzę w zapisanych postach, eksportach czatów czy plikach o nazwach będących świadectwem desperacji: „nowe_final_ostateczne_poprawione2.docx". Osobisty system semantyczny ma uporządkować właśnie ten świat – nie idealny katalog bibliotekarza, lecz realną pamięć użytkownika.

SQLite z rozszerzeniem vss jako lokalny warsztat pamięci semantycznej

SQLite jest w tym projekcie wyborem programistycznie trzeźwym. Nie wymaga ciężkiej infrastruktury serwerowej, działa lokalnie i mieści całą bazę w jednym pliku; jest powszechnie znany oraz wystarczająco stabilny dla większości osobistych zastosowań. W czasach, gdy każdą notatkę chce się natychmiast wysłać do chmury, a każde kliknięcie obłożyć subskrypcją, lokalny plik SQLite brzmi niemal jak akt obywatelskiego oporu. Nie dlatego, że chmura jest z natury zła, lecz ponieważ nie każdy problem wymaga wynajęcia cyfrowego pałacu, gdy wystarczy dobrze zorganizowany warsztat w garażu.

Aby SQLite mógł obsługiwać wyszukiwanie podobieństwa, Borwankar wykorzystuje rozszerzenie sqlite-vss. System ten wymaga wczytania dwóch modułów: vector0, dostarczającego typy i funkcje, oraz vss0, odpowiadającego za wirtualną tabelę wyszukiwania. Jest to kluczowe, gdyż klasyczny SQLite sam w sobie nie jest bazą wektorową – dopiero rozszerzenie nadaje mu zdolność korzystania z indeksów podobieństwa opartych na FAISS. Mamy tu do czynienia z modelem rozbudowy starej, sprawdzonej infrastruktury o nowe kompetencje semantyczne. To nie rewolucja poprzez spalenie biblioteki, lecz przez dobudowanie do niej inteligentnego katalogu.

Sercem tej architektury jest wzorzec dwóch tabel. Pierwsza, klasyczna, przechowuje właściwą treść posta, autora, datę, nazwę subreddita, punktację i inne metadane. Druga, wirtualna, przechowuje indeksy wektorowe. Obie są powiązane za pomocą identyfikatora rowid. Ten pozornie techniczny szczegół jest w rzeczywistości lekcją pokory wobec baz danych: jeśli zniszczymy spójność między tabelą treści a tabelą indeksu, wyszukiwarka zacznie zwracać błędne wyniki lub całkowicie przestanie działać. W systemach AI wiele katastrof nie przypomina wybuchu; wygląda raczej jak cicha utrata powiązania. Dlatego Borwankar kładzie szczególny nacisk na sposób aktualizacji rekordów.

Użycie instrukcji REPLACE może bowiem usunąć stary wiersz i stworzyć nowy z innym rowid, co nieuchronnie zerwie więź między główną tabelą a indeksem wektorowym. Zamiast tego należy stosować UPSERT (INSERT ... ON CONFLICT DO UPDATE), aby zachować tożsamość rekordu podczas aktualizacji. Ten fragment ma wartość niemal symboliczną.

Preprocessing jako filtr szumu semantycznego

To pokazuje, że systemy semantyczne nie są zwolnione z klasycznej dyscypliny integralności danych. Przeciwnie: im bardziej zaawansowana warstwa AI, tym dotkliwiej odczuwalne stają się błędy na poziomie fundamentu. Rurociąg rozpoczyna się jednak wcześniej – już w momencie pobierania danych. W projekcie redditowym służy do tego PRAW (Python Reddit API Wrapper). Pobieranie przez API wymaga respektowania limitów zapytań, obsługi przerw, wznowień i ewentualnych błędów. To kluczowy etap, ponieważ źródło danych nie jest biernym magazynem, lecz zewnętrznym systemem z własnymi ograniczeniami i regułami. Dobra architektura nie zakłada, że świat będzie posłusznie podawał dane na srebrnej tacy. Dobra architektura wie, że świat kaszle, limituje, zwraca błędy, usuwa posty i czasem odpowiada: spróbuj później, człowieku, mam swoje życie.

Po pobraniu danych następuje preprocessing, czyli czyszczenie i przygotowanie tekstu. To etap często lekceważony, a w rzeczywistości decydujący. Model embeddings nie pracuje na „czystym

sensie”, lecz na tym, co mu dostarczymy. Jeżeli tekst jest zaśmiecony linkami, artefaktami platformy, znacznikami Markdown czy fragmentami usuniętych komentarzy, embedding zakoduje nie tylko znaczenie, ale i bałagan. W efekcie system będzie ten bałagan z pełną powagą porównywał z innym bałaganem. Matematyka nie obraża się na śmieci. Ona je po prostu precyzyjnie przetwarza.

Szczególnie istotna jest zasada neutralizowania adresów URL. Borwankar nie zaleca ich bezmyślnego usuwania, lecz zastępowanie stałym tokenem, na przykład [URL]. To subtelna decyzja. Obecność linku może mieć znaczenie semantyczne: post z odnośnikiem może być poradnikiem, źródłem, komentarzem do artykułu czy rekomendacją narzędzia. Sam surowy adres internetowy – ze wszystkimi parametrami i ścieżkami – mógłby jednak zdominować reprezentację wektorową i wprowadzić szum. Token [URL] zachowuje informację o funkcji linku, usuwając jednocześnie techniczną nadmiarowość. To dobra metafora całego preprocessingu: nie amputować sensu, lecz usuwać hałas. Podobnie działa eliminacja artefaktów specyficznych dla Reddita, takich jak znaczniki [deleted], [removed], linki do użytkowników czy elementy Markdown.

Każda platforma ma swój dialekt techniczny. Jeśli chcemy budować przenośne reprezentacje semantyczne, musimy odróżnić treść od śladu interfejsu. W przeciwnym razie model zacznie uczyć się nie tylko tego, o czym ludzie piszą, ale także brudu po narzędziu, którym piszą. To trochę tak, jakby historyk badał listy miłosne, wyciągając połowę wniosków z plam po kawie na skanerze. Kolejnym krokiem jest filtrowanie jakości. Nie każdy post zasługuje na wektoryzację; bardzo krótkie wpisy czy komunikaty technicznie puste mogą pogorszyć jakość bazy. Quality filtering jest więc nie tylko optymalizacją kosztów, ale decyzją epistemiczną: do pamięci systemu trafia tylko to, co posiada minimalną wartość informacyjną.

Oczywiście każda selekcja niesie ryzyko. Można odrzucić krótki, lecz cenny wpis lub zachować długi, a pusty wywód. Nie istnieje filtr doskonały – jest tylko filtr jawny, testowalny i dostosowany do celu. Po oczyszczeniu tekstu następuje wektoryzacja. W notatce źródłowej wskazano model all-MiniLM-L6-v2 z rodziny SentenceTransformers, generujący 384-wymiarowe wektory. To model lekki, szybki i praktyczny, szczególnie użyteczny w lokalnych systemach eksperymentalnych oraz osobistych wyszukiwarkach wiedzy.

Hybrydowe wyszukiwanie łączy semantykę z twardymi filtrami

Nie każdy projekt wymaga największego możliwego modelu; czasem mniejszy, sprawny i tani obliczeniowo wariant daje lepszy bilans użyteczności niż potężny model, który dusi sprzęt, budżet i

cierpliwość. W technice, podobnie jak w polityce, rozmiar bywa mylony ze skutecznością, a potem wszyscy udają zdziwienie, że kolos ma zadyszkę.

Wektoryzacja powinna odbywać się wsadowo. To jedna z praktycznych maksym Borwankara: nie przetwarzaj tekstów pojedynczo, jeśli możesz robić to paczkami. Batch processing wykorzystuje zasoby efektywniej i znacząco przyspiesza cały proces. Ta zasada brzmi prosto, lecz oddziela prototyp od systemu, który można realnie zasilić tysiącami lub dziesiątkami tysięcy dokumentów. W świecie AI wiele różnic jakościowych zaczyna się od pozornie nudnej wydajności. Kto nie potrafi sprawnie przetworzyć danych, ten nie zbuduje inteligencji, tylko prezentację o inteligencji.

Gdy dane zostaną zapisane w bazie i zindeksowane wektorowo, pojawia się kwestia zapytań hybrydowych. Użytkownik rzadko szuka jedynie treści „podobnych semantycznie". Częściej potrzebuje precyzji: „znajdź podobne semantycznie, ale tylko z danego subreddita", „tylko po określonej dacie", „tylko wpisy z wysoką punktacją", „tylko od konkretnego autora" czy „wyłącznie w kontekście uczenia maszynowego". Wymaga to połączenia wyszukiwania wektorowego z klasycznymi filtrami metadanych.

W tym miejscu sqlite-vss ujawnia swoje ograniczenie: operacje wektorowe i standardowe filtry WHERE nie zawsze dają się elegancko połączyć w jednym kroku wykonywanym bezpośrednio przez silnik. Borwankar stosuje więc wzorzec overfetch-then-filter: pobierz z nadmiarem, następnie filtruj. System najpierw wyszukuje po wektorach większą liczbę kandydatów, niż ostatecznie ma pokazać użytkownikowi. Jeśli zapytanie zakłada dziesięć wyników, system może pobrać ich sto lub więcej.

Następnie identyfikatory tych wyników są łączone z główną tabelą postów, a klasyczny SQL filtruje je według metadanych. Dopiero w ostatnim etapie system zwraca końcową listę najlepiej dopasowanych treści. Rozwiązanie to jest praktyczne, choć nieidealne – jego skuteczność zależy od tego, czy nadmiarowo pobrana pula kandydatów okaże się wystarczająco szeroka, by po filtrowaniu pozostały trafne wyniki.

Wzorzec ten jest pouczający, gdyż obnaża istotę hybrydowości. Wyszukiwanie wektorowe wskazuje: „to jest podobne znaczeniowo", podczas gdy SQL stwierdza: „to spełnia twarde warunki". Dopiero razem tworzą użyteczną odpowiedź. Sam sens bez filtrów bywa niebezpiecznie szeroki, a same filtry bez sensu – martwo literalne. Ich połączenie pozwala zapytać: „pokaż mi nie byle co podobnego, ale podobnego w granicach, które mają dla mnie znaczenie". Przypomina to ludzką rozmowę: inteligentny rozmówca nie tylko kojarzy tematy, ale rozumie, które z tych kojarzeń są w danej sytuacji dopuszczalne.

Lokalna baza danych gwarantuje autonomię i prywatność wiedzy

Osobisty system wiedzy oparty na SQLite ma jeszcze jedną, kluczową zaletę: prywatność. Dane pozostają lokalne, co eliminuje konieczność wysyłania archiwów, zapisanych rozmów, notatek czy dokumentów do zewnętrznego dostawcy. W epoce, w której dane prywatne są traktowane jak ropa, tylko bardziej gadatliwa, lokalność staje się nie tylko kwestią wygody, ale przede wszystkim autonomii. Oczywiście taki system ma swoje ograniczenia – wymaga odpowiedniego sprzętu, konfiguracji, dbałości o kopie zapasowe i kompetencji technicznych. W zamian użytkownik zyskuje jednak coś rzadkiego: własną pamięć, która nie staje się automatycznie cudzym aktywem.

Ta prywatność ma również wymiar instytucjonalny. Organizacje, kancelarie, laboratoria czy zespoły badawcze często dysponują wiedzą wrażliwą, roboczą lub strategiczną, której nie powinny przekazywać do komercyjnych chmur. Lokalna baza semantyczna może tu pełnić funkcję wewnętrznej pamięci organizacji, umożliwiając sprawne wyszukiwanie dawnych argumentów, uchwał, analiz czy raportów. W tym ujęciu wektorowa baza danych przestaje być tylko technologiczną nowinką AI, a staje się elementem higieny instytucjonalnej.

Należy jednak pamiętać, że lokalny system semantyczny nie jest wartościowy samym faktem bycia lokalnym. Może być źle zaprojektowany, błędnie zasilony lub niedbale indeksowany. Może dawać złudzenie panowania nad wiedzą, podczas gdy w rzeczywistości będzie jedynie przeszukiwał chaotyczne archiwum. Prywatność nie zastępuje jakości, a autonomia nie zastępuje metodologii – fakt, że trzymamy bałagan na własnym dysku, nie czyni go porządkiem. Domowy strych też jest lokalny, ale znalezienie tam dokumentu sprzed pięciu lat nadal bywa archeologią ekstremalną.

Dlatego projekt Borwankara ma wartość dydaktyczną: pokazuje, że budowa osobistej pamięci AI wymaga pełnego łańcucha odpowiedzialności. Należy przemyśleć każdy etap: od źródła danych i sposobu ich pobierania, przez czyszczenie i filtrowanie jakości, model embeddings i batch processing, aż po strukturę tabel, powiązanie rowid, indeks wektorowy, strategię aktualizacji, overfetch-then-filter i sposób prezentacji wyników. Każde z tych ogniw jest jak element łańcucha dowodowego – jeśli jedno zawiedzie, system będzie produkował jedynie pozory trafności.

Projekt Redditowy jest w tym ujęciu małą szkołą budowy RAG-u, która poprzedza samo wdrożenie LLM. Uczy, że model językowy to zaledwie ostatni aktor w procesie. Przed nim musi działać pamięć, przed pamięcią indeks, przed indeksem embeddings, a przed nimi czysty tekst i rozsądne pobranie danych wynikające ze zrozumienia celu. Kto zaczyna od wyboru modelu LLM zamiast od

architektury wiedzy, przypomina człowieka, który kupuje megafon, zanim ustalił, czy ma coś prawdziwego do powiedzenia.

Od osobistej pamięci do rygoru naukowej infrastruktury

System oparty na Reddit i SQLite uwidacznia różnicę między wyszukiwaniem publicznym a osobistym. Publiczna wyszukiwarka internetowa odpowiada na pytanie: co znajduje się w sieci? Osobista wyszukiwarka semantyczna pyta natomiast: co znajduje się w mojej pamięci, moich zapisach, moim korpusie i moim świecie pracy?

To jakościowo inny problem. Dla badacza tworzącego artykuły, petycje, analizy czy strategie, własne archiwum bywa cenniejsze niż kolejna ogólna odpowiedź z sieci. W epoce nadmiaru informacji przewagę daje nie dostęp do wszystkiego, lecz zdolność odnalezienia właściwej rzeczy we własnym zasobie. Różnica ta staje się kluczowa w kontekście lokalnych LLM. Sam model językowy, uruchomiony na przykład przez Ollama, posiada ograniczoną wiedzę i nie zna prywatnych dokumentów użytkownika. Dopiero połączenie go z lokalną bazą wektorową tworzy prywatnego asystenta, który operuje na konkretnym korpusie danych.

RAG przestaje być wtedy jedynie techniką redukcji halucynacji. Staje się architekturą osobistej suwerenności poznawczej: model nie mówi dlatego, że „wie wszystko", lecz ponieważ przed udzieleniem odpowiedzi otrzymał właściwe fragmenty naszej wiedzy. W tym miejscu pojawia się jednak granica projektu SQLite. Dla osobistych archiwów składających się z dziesiątek tysięcy wpisów i notatek, pojedynczy plik SQLite jest rozwiązaniem eleganckim. Jednak przy większej skali, w pracy naukowej, przy wielu użytkownikach, złożonych filtrach czy głębokich relacjach między dokumentami, niezbędna jest cięższa infrastruktura. Wtedy Borwankar przechodzi do PostgreSQL i pgvector – świata, w którym klasyczna relacyjna baza danych zyskuje natywną zdolność przechowywania i przeszukiwania wektorów. To przejście nie jest kapitulacją prostoty, lecz uznaniem skali problemu.

Projekt Redditowy służy więc jako pierwsze laboratorium: uczy lokalności, czyszczenia chaosu i pragmatyki SQLite. Projekt ArXiv będzie laboratorium drugim – bardziej naukowym, strukturalnym i wymagającym. Tam dane nie będą już luźnymi postami, lecz publikacjami z tytułami, autorami, abstraktami i detalami ukrytymi w PDF-ach. Pytanie nie będzie brzmiało jedynie: „co przypomina moje zapytanie?", lecz: „które prace mówią o tej metodzie, które abstrakty wskazują kierunek, a które fragmenty sekcji zawierają szczegół techniczny?". To wymaga wyszukiwania wielopoziomowego – nie po prostu pamięci, lecz pamięci z anatomią. ArXiv, PostgreSQL i pgvector.

Od osobistej pamięci do naukowej maszyny wyszukiwania sensu. Po projekcie Redditowym widać, jak z chaotycznych treści można zbudować lokalną pamięć semantyczną. Nauka jednak stawia poprzeczkę wyżej. Publikacja naukowa nie jest postem ani krótką notatką; jest strukturą argumentacyjną. Posiada tytuł, abstrakt, metodologię, wyniki, bibliografię i równania matematyczne, które wyglądają jak język obcego plemienia, dopóki nie zrozumiemy ich funkcji. Wyszukiwanie naukowe nie może więc ograniczać się do sprawdzania obecności słowa; musi pytać, gdzie w strukturze wiedzy znajduje się szukany sens.

Tradycyjna wyszukiwarka słów kluczowych jest w nauce potrzebna, lecz niewystarczająca. Ten sam problem badawczy może być opisywany różnymi terminami, a ten sam algorytm występować pod odmiennymi etykietami. Autorzy korzystają z różnych tradycji terminologicznych czy wariantów matematycznych właściwych dla ich subdyscypliny. Kto szuka wyłącznie dosłownej frazy, łatwo przeoczy prace mówiące o tym samym problemie innym językiem. W nauce różnica języka nie zawsze oznacza różnicę treści – czasem to tylko kwestia innego seminarium, szkoły lub grantu. Tak, nauka też ma mody; tylko ubiera je w poważniejsze marynarki.

Właśnie dlatego Borwankar przechodzi od systemu SQLite do naukowej wyszukiwarki ArXiv opartej na PostgreSQL i pgvector. System ten pobiera literaturę z ArXiv, wykorzystując relacyjną bazę rozszerzoną o wektory. To istotny wybór architektoniczny. PostgreSQL nie jest modną zabawką z laboratorium AI, lecz dojrzałą bazą znaną ze spójności i elastyczności. Dodanie pgvector nie unieważnia tej tradycji, lecz ją rozszerza: sens zostaje dołączony do struktury, a nie wrzucony w cyfrowy chaos.

Różnica między SQLite-vss a PostgreSQL z pgvector to różnica skali i rygoru. SQLite jest znakomitym narzędziem dla osobistej pamięci; PostgreSQL natomiast dominuje tam, gdzie wymagana jest odporność, złożone relacje, kontrola integralności i profesjonalna obsługa danych.

Wyszukiwanie hybrydowe łączy dyscyplinę metadanych z elastycznością semantyczną

W projekcie ArXiv nie chodzi jedynie o zapisanie fragmentów tekstu i wektorów, lecz o zbudowanie bazy wiedzy pozwalającej łączyć twarde metadane z wyszukiwaniem semantycznym: autora z podobieństwem, datę z metodą, kategorię z problemem, a abstrakt z konkretnym fragmentem sekcji. To połączenie stanowi sedno wyszukiwania hybrydowego. Relacyjna baza danych zapewnia dyscyplinę – daty, identyfikatory, autorów, zależności, klucze obce i filtry. Warstwa wektorowa

wnosi natomiast elastyczność semantyczną: podobieństwo, sąsiedztwo pojęć i zdolność odnajdywania tekstów bliskich znaczeniowo, nawet przy braku identycznych słów.

Dopiero razem tworzą instrument, którego potrzebuje badacz. Sam SQL bywa zbyt literalny, sama baza wektorowa – zbyt miękka. Hybryda pozwala sformułować zapytanie: „znajdź prace po 2020 roku z tej dziedziny, ale nie według konkretnego słowa, lecz według sensu pytania". To jest różnica między szukaniem igły w stogu siana a zapytaniem, dlaczego w ogóle tyle tu siana. Kluczowym pomysłem Borwankara jest wielopoziomowe wyszukiwanie, wynikające z faktu, że publikacja naukowa ma warstwy. Abstrakt określa ogólny zarys pracy, sekcja metodologiczna opisuje sposób wykonania badania, wyniki prezentują rezultaty, a dyskusja interpretację tychże wyników. Wniosek może zawierać syntetyczne twierdzenie, podczas gdy szczegół techniczny często pozostaje ukryty wewnątrz tekstu. Wyszukiwanie po abstrakcie pozwala odnaleźć szeroki krajobraz badawczy; wyszukiwanie po chunkach z wnętrza dokumentu prowadzi do konkretnych detali. Oba poziomy są niezbędne, bo nauka ma mapę i mikroskop.

Źródło opisuje ten projekt poprzez dwie główne tabele: papers oraz paper_sections (lub paper_chunks). Tabela papers przechowuje identyfikatory, tytuły, autorów, daty oraz wektor abstraktu. Tabela sekcji lub chunków zawiera pocięte fragmenty wnętrza publikacji wraz z ich osobnymi wektorami. Ta architektura jest kluczowa: nie istnieje jeden wektor, który „załatwiłby" całą publikację, gdyż streszczenie i detal pełnią różne funkcje poznawcze. Wektor abstraktu pomaga odnaleźć prace tematycznie powiązane, natomiast wektor fragmentu pozwala dotrzeć do precyzyjnej metody, równania, warunku eksperymentalnego, nazwy algorytmu czy ograniczenia badania.

Taki podział poziomów chroni przed jednym z częstych błędów systemów RAG: przed spłaszczeniem dokumentu. Potraktowanie długiej publikacji jako jednolitej masy tekstu grozi zgubieniem jej wewnętrznej logiki, zaś zbyt drobne cięcie – utratą kontekstu. Z kolei wyszukiwanie wyłącznie w abstraktach daje ogólną tematykę bez szczegółów, a szukanie tylko w chunkach dostarcza detali kosztem obrazu całości. Efektywne wyszukiwanie naukowe musi zatem działać jak kompetentny badacz: najpierw rozpoznaje pole, następnie analizuje fragmenty i wraca do struktury argumentu.

Rurociąg pobierania danych rozpoczyna się od API ArXiv, z którego system pobiera metadane, linki do PDF-ów oraz informacje o autorach, datach i kategoriach. Już na tym etapie niezbędna jest dyscyplina: poszanowanie limitów zapytań, obsługa błędów, powtarzalność procesu i możliwość wznowienia pobierania. Następnie pliki PDF są przetwarzane za pomocą narzędzi do ekstrakcji tekstu – w notatce wskazano PyMuPDF. Jest to etap znacznie trudniejszy, niż mogłoby się wydawać.

Wyzwania ekstrakcji i precyzja chunkingu danych

PDF jest formatem prezentacyjnym, a nie logicznym zapisem wiedzy. Dokument może wyglądać przejrzyście dla człowieka, lecz dla maszyny bywa labiryntem kolumn, nagłówków, stopek, numerów stron czy podpisów pod rysunkami i przerwanych zdań. Ekstrakcja tekstu z publikacji naukowych to coś więcej niż proste „kopiuj-wklej". Układy wielokolumnowe często zaburzają kolejność zdań, a powtarzające się na każdej stronie nagłówki i stopki generują zbędny szum. Bibliografia wprowadza długie ciągi nazwisk – istotne jako aparat naukowy, lecz niekoniecznie powinny one dominować w embeddingu. Równania bywają odczytywane jako chaotyczne sekwencje znaków, tabele tracą strukturę, a wykresy i schematy często pozostają niewidoczne dla klasycznego parsera tekstowego. Właśnie dlatego w notatce źródłowej pojawia się wątek modeli multimodalnych: przyszłe systemy będą musiały rozumieć nie tylko tekst, ale i obrazy, kod czy tabele.

Po ekstrakcji następuje czyszczenie tekstu. Należy usunąć artefakty, przywrócić logiczny porządek i rozpoznać sekcje, by przygotować materiał do chunkingu. W naukowym RAG-u podział na fragmenty ma większe znaczenie niż w prostym archiwum notatek, gdyż publikacje zawierają gęstą argumentację. Notatka sugeruje okna od 512 do 1024 tokenów (mniej więcej jeden do trzech akapitów) z zakładką na poziomie około 20 procent.

Ta zakładka nie jest stratą miejsca – jest ubezpieczeniem sensu. Dzięki niej fragmenty leżące na granicy dwóch chunków nie zostają brutalnie rozdzielone. Warto się przy tym zatrzymać, gdyż chunking to jeden z najbardziej niedocenianych elementów systemów RAG. Zbyt długi fragment może obejmować zbyt wiele tematów, co rozmywa embedding. Zbyt krótki zaś pozbawia nas kontekstu, czyniąc odpowiedź fragmentaryczną. Chunk pocięty mechanicznie w połowie zdania przypomina świadek wyrwany z rozmowy: coś mówi, ale trzeba zgadywać, do czego się odnosi. Precyzyjne cięcie tekstu jest więc formą interpretacji struktury dokumentu. Nie ma w tym romantyzmu, jest za to solidna praca, dzięki której model nie będzie musiał udawać, że rozumie fragment bez głowy i ogona.

Wektoryzacja abstraktów i chunków odbywa się osobno. W notatce wskazano model all-MiniLM-L6-v2 generujący 384-wymiarowe embeddingi. To wybór pragmatyczny: model jest lekki, szybki i wystarczający do wielu zadań semantycznych. Można stosować większe modele specjalistyczne lub domenowe, jednak zwiększa to koszt, złożoność i zapotrzebowanie na zasoby. Tu powraca żelazne prawo infrastruktury AI: każdy zysk jakości należy zestawić z kosztem obliczeń, pamięci i utrzymania. Największy model nie zawsze jest najlepszym systemem; czasem okazuje się jedynie najdroższą odpowiedzią na źle postawione pytanie.

Po zapisaniu wektorów w PostgreSQL z pgvector można zastosować indeksy przyspieszające wyszukiwanie.

Precyzyjny retrieval i lokalność jako gwaranty odpowiedzialności

W notatce wskazano na HNSW oraz operator odległości kosinusowej. HNSW pozwala uniknąć pełnego skanowania wszystkich rekordów, co przy dużych zbiorach publikacji byłoby zbyt wolne. W praktyce wyszukiwanie naukowe wymaga bowiem nie tylko trafności, lecz także responsywności. Badacz, który czeka wieczność na każde zapytanie, szybko odkryje, że cierpliwość jest cnotą piękną, ale niekoniecznie skalowalną.

Operator podobieństwa lub odległości pełni tu funkcję bramy do przestrzeni semantycznej. Zapytanie użytkownika zostaje zamienione na wektor tym samym modelem embeddings, który wcześniej wektoryzował dokumenty. Następnie baza porównuje wektor pytania z wektorami abstraktów i fragmentów. Wyniki mogą być filtrowane według metadanych – na przykład daty publikacji, autora lub kategorii – oraz ograniczane progiem podobieństwa. Przykładem jest odrzucanie wyników o wartości niższej niż 0.7, aby zmniejszyć ryzyko halucynacyjnych dopasowań. Sam próg musi być oczywiście empirycznie dostosowany do domeny i modelu. Nie jest objawieniem z góry, tylko parametrem do testowania.

Wyszukiwanie wielopoziomowe powinno działać równolegle: osobno po abstraktach, a osobno po sekcjach lub fragmentach. Wyniki z abstraktów dają orientację, które prace są tematycznie istotne, natomiast wyniki z fragmentów wskazują konkretne miejsca, które mogą zawierać odpowiedź. Dzięki temu system może odpowiedzieć nie tylko: „ta praca jest podobna do zapytania”, ale także: „w tej pracy, w tym fragmencie, znajduje się szczegół odpowiadający pytaniu”.

To zasadnicza różnica. W nauce nie wystarczy znaleźć publikacji; trzeba wiedzieć, gdzie dokładnie znajduje się argument. Następnie następuje etap konstrukcji kontekstu dla LLM. Najlepiej dopasowane abstrakty i fragmenty sekcji tworzą zestaw dowodów, który zostaje wstrzyknięty do promptu lokalnego modelu językowego generującego odpowiedź. W notatce źródłowej podkreślono, że prompt powinien nakazywać modelowi używanie wyłącznie dostarczonych materiałów oraz dopisywanie cytowań do każdego faktu. To nie jest biurokratyczna fanaberia, lecz mechanizm kontroli epistemicznej. Model językowy bez cytowań jest mówcą; model zmuszony do pracy na źródłach zaczyna przypominać asystenta badawczego.

Różnica ta jest jak między toastem a ekspertyzą. W naukowym RAG-u cytowanie ma szczególne znaczenie, gdyż LLM potrafi pisać płynnie o wszystkim, również o tym, czego nie wie. W świecie literatury naukowej taka biegłość jest groźna – fałszywe twierdzenie zapisane językiem akademickim wygląda jak wiedza, dopóki ktoś nie zapyta o źródło. Dlatego dobry system RAG powinien nie tylko odpowiadać, ale i pokazywać pochodzenie informacji. W przeciwnym razie zamieniamy asystenta naukowego w generator recenzji z kosmosu: elegancki ton, brak odpowiedzialności, bibliografia z równoległego wymiaru.

Istotna jest również lokalność rozwiązania. Borwankar wskazuje możliwość użycia modelu lokalnego przez Ollama zamiast wysyłania danych do chmurowych usług. W kontekście laboratoriów, firm czy kancelarii ma to kluczowe znaczenie – nie wszystkie dokumenty mogą być publiczne i nie każda notatka robocza powinna opuszczać bezpieczne środowisko użytkownika. Lokalny RAG nie jest więc tylko rozwiązaniem technicznym, lecz polityką kontroli nad własnym korpusem wiedzy.

Oczywiście lokalność ma swoją cenę. Model może być słabszy od gigantów chmurowych: mieć mniejsze okno kontekstowe, gorsze zdolności rozumowania czy wolniejszą generację. Jednak w wielu zastosowaniach badawczych najważniejsza nie jest maksymalna elokwencja, lecz kontrola nad źródłami, powtarzalność i prywatność. Jeśli system ma bazować na dostarczonych fragmentach, a nie popisywać się literackim freestyle'em, mniejszy model z dobrze zbudowanym retrieval może okazać się bardziej użyteczny niż potężny model karmiony nieprecyzyjnym kontekstem. W świecie RAG-u retrieval bywa ważniejszy niż retoryka.

Projekt ArXiv pokazuje dodatkowo, że bazy wektorowe nie zastępują metodologii naukowej – one ją wspierają albo deformują. System może pomóc w odkrywaniu ukrytych powiązań terminologicznych czy przyspieszeniu przeglądu literatury, ale nie zastąpi krytycznej oceny jakości badań, poprawności eksperymentu, siły dowodu ani analizy konfliktów interesów.

Semantyka jako mapa potencjału, a nie dowód prawdy

Wektor wskaże, że dwa teksty są blisko. Nie powie jednak sam z siebie, że jeden prezentuje lepszą metodę, drugi opiera się na słabszej próbie, a trzeci jest elegancko napisaną pomyłką. Semantyczna bliskość nie oznacza prawdziwości. Podobieństwo nie jest dowodem, a ranking nie jest recenzją. Jeśli system wyszukuje prace podobne do zapytania, tworzy mapę potencjalnej relewantności, a nie mapę prawdy. Człowiek wciąż musi ocenić, czy odnalezione publikacje są rzetelne, aktualne i metodologicznie mocne. AI może skrócić drogę do źródeł, ale nie powinna być zwolnieniem z myślenia. W przeciwnym razie badacz zamienia się w operatora autopilota, który zapomniał, że

samoloty czasem lecą prosto w górę tylko na ekranie. Jednocześnie potencjał takiego systemu jest ogromny.

Naukowa wyszukiwarka semantyczna może pomóc początkującemu badaczowi odnaleźć obszar literatury, którego nie zna terminologicznie. Wspiera interdyscyplinarność, gdyż różne dziedziny często opisują podobne problemy innymi słowami; pozwala wykrywać analogie między ekonomią, informatyką, biologią, teorią sieci a naukami społecznymi. Może odnajdywać metody stosowane w jednej dyscyplinie, które nadają się do transferu do innej. To jedna z najciekawszych funkcji embeddings: potrafią one czasem mostkować języki dziedzin, zanim zrobią to ludzie. W tym miejscu system ArXiv staje się czymś więcej niż wyszukiwarką – staje się narzędziem reorganizacji wiedzy. Klasyczna bibliografia prowadzi po śladach cytowań, wyszukiwarka słów kluczowych po śladach terminów, a semantyczna po śladach podobieństwa znaczeń. Każdy z tych trybów ma inną epistemologię. Cytowania ukazują sieć uznania i zależności akademickich, choć mogą utrwalać hierarchie. Słowa kluczowe wskazują deklarowane pojęcia, lecz nie zawsze chwytają sens. Wektory pokazują bliskość semantyczną, ale mogą ukrywać powody tej bliskości.

Dobra nauka powinna korzystać ze wszystkich trzech metod, nie koronując żadnej na króla absolutnego. Dlatego tak istotne jest wyszukiwanie hybrydowe. Łączenie BM25, metadanych i embeddings pozwala zniwelować słabości każdego z tych podejść. Słowa kluczowe są niezbędne, gdy termin techniczny jest kluczowy i nie może zostać rozmyty. Wektory pomagają, gdy pojęcia są synonimiczne, rozproszone lub interdyscyplinarne. Metadane natomiast pozwalają ograniczyć wyniki do konkretnego czasu, autora, kategorii czy typu publikacji. Hybryda jest mniej efektowna niż obietnica jednej cudownej metody, ale za to bardziej prawdziwa. A prawda rzadko przychodzi z fajerwerkami – częściej z dokumentacją.

Projekt PostgreSQL-pgvector ma istotne znaczenie dla przyszłości instytucjonalnych baz wiedzy. Wyobraźmy sobie archiwum ekspertyz, orzeczeń, petycji, raportów, opinii prawnych czy analiz ekonomicznych. Klasyczne wyszukiwanie pozwoli znaleźć nazwisko, datę lub frazę. Wyszukiwanie semantyczne umożliwi odnalezienie wcześniejszego argumentu, podobnego problemu, analogicznej struktury sporu, pokrewnej luki prawnej lub przykładu zastosowania tej samej koncepcji. W instytucji myślącej długofalowo taka pamięć jest bezcenna – nie dlatego, że zastępuje ekspertów, lecz chroni ich przed powtarzaniem własnego zapomnienia.

Właśnie tu ujawnia się społeczny wymiar baz wektorowych. Organizacje rzadko cierpią na brak danych; częściej brakuje im pamięci operacyjnej. Wiedza istnieje, ale jest rozproszona. Ktoś przygotował analizę, stanowisko lub zebrał argumenty, lecz plik utknął w folderze, mailu, archiwum albo w głowie osoby, która już pracuje gdzie indziej. Baza wektorowa może stać się narzędziem

wydobywania tej wiedzy z instytucjonalnego mułu. Nie wygeneruje mądrości automatycznie, ale zapobiegnie temu, by zginęła ona pod nazwą „wersja finalna poprawiona nowa naprawdę ostatnia".

Tak rozumiany system ArXiv jest modelem dla szerszej klasy rozwiązań. To przykład dobrze określonego korpusu, lecz podobną architekturę można zastosować do orzecznictwa, dokumentacji technicznej, aktów prawnych czy repozytoriów kodu. Kluczowe pozostaje to samo: rozpoznanie struktury dokumentu, zachowanie metadanych, sensowne cięcie tekstu, tworzenie embeddings i wymuszanie źródłowej odpowiedzialności generowanych odpowiedzi. W naukowym RAG-u szczególnie ważne jest rozróżnienie między wyszukaniem a syntezą. Retrieval odpowiada za znalezienie fragmentów, LLM zaś za ich językowe złożenie. Jeśli retrieval jest słaby, model będzie syntetyzował przypadkowe treści. Jeśli prompt jest zbyt luźny, AI może dodać własne „upiększenia". Z kolei przy rygorystycznym prompcie i ubogim kontekście odpowiedź będzie poprawna, lecz niewystarczająca. Dobry system powstaje tylko wtedy, gdy wszystkie elementy łańcucha są dostrojone. AI nie jest pojedynczym mózgiem, lecz orkiestrą – a źle nastrojona orkiestra potrafi zagrać fałsz w wysokiej rozdzielczości. System ArXiv prowadzi nas więc naturalnie do architektury RAG jako całości.

W projekcie Redditowym widzieliśmy lokalną pamięć osobistą, w ArXiv – naukową pamięć strukturalną. W obu przypadkach wektory służą do odnajdywania znaczących fragmentów. Jednak dopiero połączenie retrieval z modelem językowym tworzy system zdolny odpowiadać w języku naturalnym, streszczać i syntetyzować. W tym miejscu baza wektorowa przestaje być wyłącznie wyszukiwarką, a staje się zapleczem epistemicznym dla modelu LLM. To przejście jest decydujące.

RAG jako architektura odpowiedzialności i uziemienia wiedzy

LLM bez RAG-u przypomina elokwentnego rozmówcę z pamięcią zamrożoną w czasie i skłonnością do konfabulacji. RAG daje mu dostęp do aktualnych, prywatnych lub specjalistycznych źródeł, jednak dostęp ten musi być rygorystycznie kontrolowany. Model nie może odpowiadać „z siebie", jeśli ma bazować na źródłach; musi zostać zmuszony do cytowania, przyznania się do niewiedzy i trzymania kontekstu, rezygnując z kreatywności tam, gdzie niezbędna jest ścisłość. W przeciwnym razie otrzymamy nie asystenta badawczego, lecz przekonującego improwizatora, a ten w nauce przydaje się na bankiecie, nie w metodologii. Dlatego kolejny etap analizy skupi się na RAG-u: nie jako modnym skrócie, lecz jako architekturze odpowiedzialności.

Retrieval-Augmented Generation to próba odpowiedzi na problem zamrożonej, niepełnej i niespecjalistycznej wiedzy modeli językowych oraz ich podatność na halucynacje. Baza wektorowa dostarcza pamięć, embeddings – semantyczny mechanizm odnajdywania, a LLM – językową syntezę. Prompt wprowadza dyscyplinę, zaś cytowania zapewniają kontrolę. Dopiero taka całość zasługuje na miano systemu wiedzy, a nie tylko generatora tekstu w garniturze. RAG to model językowy podłączony do pamięci; sposób na uziemienie elokwencji w źródłach.

Duże modele językowe mają osobliwą właściwość: potrafią mówić tak, jakby wiedziały więcej, niż w rzeczywistości wiedzą. Ich zdolność generowania płynnego tekstu jest imponująca, lecz właśnie ta płynność bywa epistemicznie niebezpieczna. Człowiek słyszy składnię pewności, rytm ekspertyzy, styl akademicki i poprawną terminologię, przez co łatwo myli formę wypowiedzi z jej wiarygodnością. Model może złożyć zdanie brzmiące jak fragment podręcznika, choć w istocie jest ono jedynie elegancką rekonstrukcją prawdopodobieństwa, a nie wynikiem weryfikacji faktu. To nie drobna wada, lecz centrum problemu.

Borwankar ujmuje tę sprawę praktycznie: wiedza LLM zostaje zamrożona w momencie zakończenia treningu. Model nie zna najnowszych informacji, prywatnych dokumentów użytkownika, wewnętrznych archiwów organizacji ani niszowych danych specjalistycznych, jeśli nie zostały mu dostarczone. Może rozpoznawać wzorce językowe i struktury argumentacji, ale nie ma bezpośredniego dostępu do konkretnego korpusu, na którym użytkownik chce oprzeć rozumowanie. Właśnie dlatego RAG zostaje przedstawiony jako architektura rozwiązująca brak dostępu do aktualnych informacji oraz skłonność do halucynacji.

Retrieval-Augmented Generation, czyli generowanie wspomagane wyszukiwaniem, łączy dwie odmienne kompetencje. Pierwszą jest retrieval: odnalezienie właściwych fragmentów wiedzy w bazie danych. Drugą – generation: złożenie ich w odpowiedź w języku naturalnym. Model nie ma wtedy improwizować z pamięci treningowej, lecz odpowiadać na podstawie materiału dostarczonego w kontekście. RAG nie jest więc zwykłym „ulepszeniem promptu", lecz zmianą architektury poznawczej systemu. Model przestaje być samotnym mówcą. Staje się komentatorem akt sprawy. Ta różnica jest zasadnicza.

W zwykłej rozmowie z LLM pytanie trafia bezpośrednio do modelu, który odpowiada w oparciu o swoje parametry i ogólne reprezentacje wiedzy. W systemie RAG zapytanie najpierw uruchamia proces wyszukiwania: zostaje zamienione na wektor przy użyciu modelu embeddings, a baza wektorowa odnajduje fragmenty semantycznie podobne do pytania. Następnie są one montowane w kontekst wraz z instrukcją dla LLM. Dopiero wtedy model generuje odpowiedź. Zanim więc zacznie mówić, system każe mu przeczytać właściwe kartki. To właśnie jest uziemienie – grounding.

Nie oznacza ono pełnej gwarancji prawdy, gdyż żaden system informatyczny nie eliminuje całkowicie ryzyka błędu. Oznacza jednak radykalne zmniejszenie swobody konfabulacji. Model nie powinien odpowiadać na podstawie luźnych skojarzeń, lecz pobranych fragmentów. Jeśli ich brakuje, dobrze zaprojektowany prompt powinien zmusić go do przyznania: „nie wiem", „brak danych" lub „kontekst nie zawiera odpowiedzi". To jedna z najważniejszych zasad dobrego RAG-u. Model potrafiący powiedzieć „nie wiem" jest bardziej użyteczny niż taki, który na każde pytanie odpowiada jak samozwańczy ekspert – pewnie i bardzo podejrzanie.

Pięć filarów RAG i rola orkiestratora w uziemianiu wiedzy

W źródłach system RAG zostaje rozpisany na pięć komponentów: bazę wektorową, silnik osadzeń, wyszukiwanie hybrydowe, integrację z LLM oraz orkiestrator rurociągu. Każdy z tych elementów pełni odrębną funkcję. Baza wektorowa przechowuje chunki i ich embeddings, silnik osadzeń zamienia tekst na wektory, a wyszukiwanie hybrydowe łączy semantykę z filtrami i słowami kluczowymi. Za językową syntezę odpowiada LLM. Cały proces koordynuje orkiestrator: przyjmuje pytanie, tworzy embedding zapytania, odpytuje bazę, pobiera wyniki, układa kontekst, wysyła prompt do modelu i zwraca odpowiedź. Jeśli którykolwiek z tych elementów zawiedzie, system zaczyna produkować złudzenie kompetencji.

Rola orkiestratora jest kluczowa. To on decyduje o liczbie pobieranych fragmentów, ich sortowaniu, filtrowaniu po metadanych czy sposobie łączenia w prompt. On określa, czy wyniki zostaną oznaczone źródłami, czy model otrzyma instrukcję cytowania i czy może korzystać z wiedzy zewnętrznej. To on ustala temperaturę modelu oraz definiuje ścieżkę postępowania w przypadku słabych wyników retrieval.

Orkiestrator jest reżyserem tej operacji. LLM pełni rolę aktora, jednak aktor bez scenariusza potrafi ukraść przedstawienie. W systemach wiedzy nie chodzi jednak o błysk sceniczny, lecz o wierność materiałowi. Rurociąg RAG składa się z dwóch głównych faz: ingestion i querying. Faza ingestion to przygotowanie korpusu – dokumenty są dzielone na chunki, czyszczone, wektoryzowane i zapisywane w bazie. Faza querying rozpoczyna się w momencie zadania pytania przez użytkownika; system zamienia je na embedding, porównuje z wektorami chunków, pobiera najtrafniejsze fragmenty i przekazuje stworzony kontekst do modelu. Przykładem optymalizacji jest dzielenie tekstu na segmenty po około 200 słów z 50-słowową zakładką, co pozwala zachować spójność kontekstu na granicach fragmentów.

Ta zakładka wydaje się drobiazgiem tylko z pozoru. W praktyce decyduje o tym, czy model otrzyma pełną myśl, czy jedynie jej fragment. Chunking jest jednym z miejsc, gdzie technika spotyka hermeneutykę. Dokument nie jest bowiem workiem zdań – posiada strukturę. Zbyt mechaniczne cięcie tekstu może rozerwać argument, oddzielić tezę od uzasadnienia, definicję od przykładu lub warunek od wniosku. Z kolei zbyt duże fragmenty mogą rozmyć embedding, gdyż jeden wektor próbowałby reprezentować kilka tematów jednocześnie. Precyzyjny chunking to kompromis między kontekstem a dokładnością. Przypomina to przygotowanie materiału dla eksperta: jedno wyrwane zdanie może być niezrozumiałe, natomiast sto stron bez struktury czyni automatyzację bezcelową.

Wyszukiwanie w RAG-u może opierać się wyłącznie na wektorach, lecz często lepsze efekty przynosi podejście hybrydowe. Polega ono na połączeniu wyszukiwania semantycznego z tradycyjnym wyszukiwaniem słów kluczowych (np. BM25), stosując heurystykę wag 70/30: siedemdziesiąt procent dla sensu wektorowego i trzydzieści procent dla dopasowania słów kluczowych.

Hybrydowe mechanizmy kontroli i odpowiedzialności RAG

Ta proporcja nie jest prawem natury, lecz praktycznym punktem startowym. Jej sens jest jasny: wektory świetnie wychwytują podobieństwo znaczeń, ale słowa kluczowe bywają niezastąpione tam, gdzie termin techniczny, nazwa własna, sygnatura, przepis prawa czy symbol naukowy muszą zostać odnalezione dosłownie.

Hybryda stanowi antidotum na dwie skrajności. Czyste wyszukiwanie słów kluczowych może pominąć tekst opisujący problem innymi pojęciami. Z kolei samo wyszukiwanie wektorowe może wskazać fragment semantycznie bliski, ale przeoczyć krytyczny detal terminologiczny. W praktycznych systemach wiedzy potrzebujemy obu logik. Sens bez litery może odpłynąć. Litera bez sensu może skamienieć.

Skuteczne systemy RAG są mniej romantyczne niż marketing AI: nie wierzą w jedną cudowną technikę, lecz budują trafność z kilku niedoskonałych narzędzi, które wzajemnie korygują swoje wady. Kolejnym mechanizmem podnoszenia jakości jest reranking. Pierwszy etap retrieval może pobrać szeroką pulę kandydatów, którą następnie silniejszy model – na przykład cross-encoder – ponownie ocenia pod kątem trafności względem pytania i precyzyjniej porządkuje. Zaawansowane ewolucje RAG obejmują właśnie reranking, inteligentne semantyczne cięcie tekstu oraz wieloetapowe rozumowanie. Reranking jest kluczowy, ponieważ pierwszy wynik wyszukiwania

wektorowego nie zawsze stanowi najlepszy dowód dla odpowiedzi; bywa jedynie najbliższym sąsiadem w sensie geometrycznym, a nie najbardziej użytecznym fragmentem argumentacyjnym.

W systemach specjalistycznych próg podobieństwa pełni funkcję ochronną. Jeśli jest zbyt niski, model otrzyma fragmenty słabo powiązane z pytaniem i może sformułować odpowiedź pozornie trafną. Jeśli zaś będzie zbyt wysoki, system odrzuci wartościowe dane i odpowie zbyt wąsko lub odmówi odpowiedzi mimo istnienia informacji. Dobór progu zależy od domeny – inaczej ustawimy go dla wyszukiwarki memów, a inaczej dla dokumentacji technicznej, prawa, medycyny czy finansów.

Im wyższa stawka błędu, tym większa powinna być ostrożność. Maszyna nie musi mieć nerwów, ale projektant systemu powinien. Najbardziej charakterystycznym elementem rygorystycznego RAG-u jest prompt kontrolujący model. Obowiązuje tu zasada, że LLM powinien korzystać wyłącznie z dostarczonego kontekstu i cytować źródła dla każdego twierdzenia. To zasadnicza różnica między generatorem tekstu a narzędziem wiedzy. Generator może pisać efektownie; narzędzie wiedzy musi wykazać, skąd czerpie informacje. Wymóg cytowania nie jest ozdobą akademicką, lecz hamulcem bezpieczeństwa. Zmusza model do wiązania zdań z dokumentami, a użytkownikowi pozwala zweryfikować podstawę odpowiedzi.

Należy jednak zaznaczyć: sam nakaz cytowania nie wystarczy. Modele potrafią generować cytowania pozorne lub błędne, jeśli system nie dostarcza im dobrze oznaczonych fragmentów i nie waliduje wyników. Dlatego dojrzała architektura RAG powinna obejmować nie tylko instrukcję „cytuj", ale także mechanizm identyfikacji źródeł, metadane dokumentów oraz kontrolę zgodności odpowiedzi z kontekstem. Cytowanie bez audytu bywa jak pieczątka bez urzędu: wygląda poważnie, ale nie zawsze niesie odpowiedzialność.

Temperatura modelu to kolejny parametr kontroli. Przy analizie dokumentów naukowych lub prywatnych notatek zaleca się bardzo niską temperaturę (np. 0.1), która steruje losowością generacji. Im jest ona wyższa, tym większa kreatywność i wariantywność odpowiedzi. W RAG-u zazwyczaj nie szukamy kreatywności w sensie wymyślania nowych możliwości, lecz ścisłości, powtarzalności i wierności źródłom. Kreatywność ma swoje miejsce w eseju czy metaforze, ale gdy pytamy o treść dokumentu, model nie powinien zachowywać się jak poeta przy bufecie.

Nie oznacza to, że odpowiedź RAG ma być sucha jak instrukcja obsługi śrubokręta. Model może wyjaśniać, syntetyzować i porządkować, jednak jego swoboda powinna dotyczyć formy objaśnienia, a nie faktów. Może pomóc zrozumieć materiał, ale nie powinien go dopisywać. To granica między interpretacją a konfabulacją. W dobrym systemie RAG interpretacja jest jawna i oparta na źródłach; konfabulacja jest błędem – nawet jeśli brzmi ładnie. Zwłaszcza jeśli brzmi ładnie.

RAG rozwiązuje również problem aktualności. Model bazowy nie zna dokumentów opublikowanych po treningu, natomiast retrieval może pobrać najnowsze materiały z aktualizowanej bazy. W obszarach prawa, nauki czy technologii jest to kluczowe, gdyż wiedza starzeje się nierówno. Twierdzenie matematyczne pozostaje stabilne przez stulecia, ale przepis prawa czy wersja biblioteki mogą zmienić się w kilka miesięcy. RAG pozwala oddzielić kompetencję językową modelu od aktualności korpusu; zamiast przetrenowywać model przy każdej zmianie świata, wystarczy aktualizować bazę wiedzy.

Architektura ta umożliwia również pracę na danych prywatnych. LLM nie zna dokumentów użytkownika, ale system retrieval dostarcza je w kontekście. Dzięki temu można budować asystentów dla kancelarii, redakcji, laboratoriów czy firm. Warunkiem jest jednak odpowiedzialne zarządzanie dostępem. Baza wektorowa musi współpracować z uprawnieniami i politykami prywatności, by nie stworzyć semantycznej wyszukiwarki, która zbyt inteligentnie znajdzie to, czego nie powinna pokazać. W tym punkcie ujawnia się głębszy, instytucjonalny sens RAG-u: nie jest on tylko techniką poprawiania chatbota.

RAG jako architektura pamięci organizacyjnej i źródłowości

To architektura pamięci organizacyjnej. Organizacje gubią własną wiedzę, ponieważ dokumenty są rozproszone, nazwy plików przypadkowe, ludzie odchodzą, projekty się kończą, a decyzje tracą kontekst i argumentację w gąszczu maili. RAG pozwala wydobyć wcześniejsze stanowiska, analogiczne analizy, przeszłe decyzje, ukryte zależności i zapomniane uzasadnienia. Nie zastąpi on rozumu instytucji, ale może ograniczyć jej amnezję. A amnezja instytucjonalna jest kosztowna; czasem droższa niż głupota, bo udaje doświadczenie.

Jednocześnie RAG nie jest panaceum; ma swoje słabości. Może pobrać błędne fragmenty, pominąć kluczowy dokument lub źle zinterpretować kontekst i zacytować treści nieadekwatne do odpowiedzi. Ograniczeniem bywa zbyt małe okno kontekstowe lub podatność na dokumenty „zatrute” treścią – czyli prompt injection ukryty w źródłach. System może wzmacniać błędy obecne w korpusie i produkować odpowiedzi przekonujące, lecz niepełne. RAG redukuje halucynacje, ale ich nie unicestwia. Kto obiecuje pełną eliminację błędu, sprzedaje nie technologię, tylko kadzidło.

Kluczowy pozostaje problem jakości korpusu. RAG operuje na tym, co mu dostarczono. Jeśli baza zawiera dokumenty przestarzałe, sprzeczne, słabo podzielone lub niskiej jakości, model będzie syntetyzował materiał wadliwy. Zasada „garbage in, garbage out” zyskuje tu bardziej wyrafinowaną

formę: „garbage retrieved, garbage eloquently explained”. System potrafi pięknie objaśnić błąd, jeśli ten został mu podany jako kontekst. Dlatego praca nad RAG-iem nie kończy się na uruchomieniu modelu; obejmuje kurację danych, wersjonowanie, klasyfikację źródeł i mechanizmy odświeżania bazy. W zaawansowanych systemach pojawia się multihop reasoning – wieloetapowe rozumowanie wymagające pobrania informacji z kilku niezależnych źródeł.

Proste pytanie może wymagać jednego fragmentu, lecz trudne wymaga łańcucha: najpierw definicji, potem przykładu, wyjątku, aktualnego stanu i porównania. W takim scenariuszu pojedynczy przebieg retrieval często nie wystarcza. System musi iteracyjnie pobierać kolejne fragmenty, sprawdzać hipotezy i składać odpowiedź z wielu punktów oparcia. To kierunek rozwoju kluczowy dla nauki, prawa, analizy politycznej i strategii – dziedzin, w których rzadko pytamy o jedno zdanie, a częściej o układ zależności.

Wielopoziomowe wyszukiwanie, znane z projektu ArXiv, jest jedną ze sposobów radzenia sobie z tym problemem. Można najpierw wyłonić prace istotne tematycznie po abstraktach, by następnie wejść głębiej w konkretne sekcje. Podobnie w archiwum prawnym: od ogólnych spraw podobnych do konkretnych przepisów i uzasadnień orzeczeń; a w dokumentacji technicznej – od identyfikacji modułu do konkretnej funkcji. RAG nie musi być płaskim wyszukiwaniem top-k; powinien być architekturą przejścia od mapy do detalu.

W tym kontekście istotna jest rola modeli lokalnych. Borwankar postuluje uruchamianie LLM lokalnie, m.in. przy użyciu Ollama i modeli takich jak Llama 3.1 8B. Lokalny model w połączeniu z lokalną bazą wektorową tworzy system, który nie musi wysyłać prywatnych dokumentów do zewnętrznego API, co ma fundamentalne znaczenie dla prywatności, kosztów i kontroli. W rozwiązaniach komercyjnych chmura jest wygodna, ale wygoda bywa narkotykiem infrastrukturalnym. Początkowa łatwość prowadzi do rosnących rachunków, pogłębiającej się zależności i sytuacji, w której prywatna wiedza zaczyna żyć w cudzym centrum danych.

Lokalny RAG nie jest jednak technologiczną ascezą dla samej ascezy. Nie chodzi o ideologiczne odrzucenie chmury, lecz o dobór architektury do ryzyka i celu. Jeśli dane są publiczne i mało wrażliwe, a kluczowa jest jakość największego modelu, chmura może być rozsądnym wyborem. Jeśli jednak dane są strategiczne, badawcze lub prawne, lokalność staje się warunkiem zaufania. Dojrzałość nie polega na dogmatycznym wyborze między chmurą a lokalnością, lecz na pytaniach: kto widzi dane, kto kontroluje model, kto płaci rachunek i kto ponosi odpowiedzialność za błąd?

RAG zmienia także rolę użytkownika. Przestaje on być jedynie osobą zadającą pytanie chatbotowi, a staje się kuratorem korpusu, projektantem zapytań i kontrolerem źródeł. Wymaga to nowej kultury pracy z AI: zamiast zachwycać się odpowiedzią, należy pytać o pochodzenie informacji,

wystarczalność kontekstu i aktualność źródła. Trzeba sprawdzić, czy wynik retrieval nie pominął kontrargumentów, czy model odróżnił fakt od interpretacji i czy przyznał się do niewiedzy tam, gdzie powinien. Kompetencja krytyczna w epoce RAG-u nie znika – staje się bardziej techniczna.

Co najciekawsze, RAG przywraca wagę dokumentowi. W pierwszej fali fascynacji LLM wydawało się, że model jest centrum świata. RAG pokazuje, że choć model jest potężny, bez dostępu do źródeł pozostaje jedynie elokwentnym generalistą. Dokument, korpus i metadane wracają do gry, co jest dobrą wiadomością dla nauki, prawa i poważnej publicystyki. Przyszłość AI nie musi oznaczać triumfu gadania bez źródeł; może przynieść nową epokę źródłowości, jeśli zbudujemy systemy nagradzające ugruntowanie zamiast samej płynności.

Właśnie dlatego RAG prowadzi naturalnie do pytania o standaryzację. Skoro systemy wektorowe, embeddings, metryki i wyszukiwanie hybrydowe stają się kluczowe, rośnie potrzeba wspólnego języka opisu tych operacji. Dzisiejszy rynek baz wektorowych jest rozproszony: różni dostawcy, różne API i odmienne sposoby definiowania indeksów oraz zapytań.

VQL i lokalna infrastruktura jako gwarancja niezależności

Programista często zmuszony jest pisać kod zależny od konkretnego narzędzia, administrator tworzyć osobne skrypty, a badacz dostosowywać eksperymenty do wybranej platformy. Przypomina to świat sprzed pełnej dojrzałości SQL-a: wiele silników, wiele zwyczajów i brak jednej gramatyki rozmowy z danymi. W tym miejscu pojawia się koncepcja VQL (Vector Query Language), przedstawiona przez Borwankara jako propozycja eksperymentalna i teoretyczna, a nie gotowy standard. Jej sens wynika z całej dotychczasowej drogi.

Skoro wektory stały się nowym typem danych, podobieństwo nową operacją wyszukiwania, metryki odległości częścią logiki zapytań, a RAG wymaga powtarzalnych procedur retrieval, nieuchronnie pojawia się pytanie: czy potrzebujemy języka pozwalającego opisywać te operacje niezależnie od konkretnego dostawcy? Odpowiedź Borwankara jest ostrożna, lecz ambitna. VQL miałby być rodzajem SQL-a dla przestrzeni wektorowych – językiem łączącym znane klauzule SELECT, FROM, WHERE i LIMIT z operacjami typu VECTOR_OPERATION oraz USING METRIC. Nie chodzi tu o zastąpienie SQL-a, lecz o jego semantyczne rozszerzenie. Klasyczne zapytanie pytało: które rekordy spełniają warunek? Zapytanie VQL pytałoby dodatkowo: które wektory są najbliższe danemu wektorowi, według wskazanej metryki, z uwzględnieniem filtrów metadanych i progów podobieństwa?

To nie jest jedynie kosmetyczna zmiana języka, lecz próba sformalizowania nowego sposobu myślenia o danych. Jeśli RAG jest architekturą odpowiedzialności modelu wobec źródeł, VQL może stać się gramatyką odpowiedzialnego retrieval. Zamiast ukrywać wyszukiwanie podobieństwa w kodzie aplikacji czy API konkretnego dostawcy, można by opisać je deklaratywnie: co wybieramy, z jakiej kolekcji, jaką operacją wektorową, według jakiej metryki, z jakim filtrem i limitem. Taka standaryzacja ułatwiłaby audyt, przenośność, porównywanie systemów oraz edukację. Dopóki bowiem każdy dostawca mówi własnym dialektem, użytkownik łatwo staje się zakładnikiem narzędzia. A zakładnik, nawet z pięknym interfejsem, nadal pozostaje zakładnikiem.

Zanim jednak potraktujemy VQL jako projekt przyszłości, należy domknąć wątek lokalnych LLM, gdyż RAG nie jest wyłącznie techniką retrieval. Jest on również odpowiedzią na pytanie o miejsce działania inteligencji: w cudzej chmurze czy pod kontrolą użytkownika. Lokalny model, lokalna baza wektorowa i lokalny orkiestrator tworzą architekturę, która może być wolniejsza, skromniejsza i mniej spektakularna niż największe systemy komercyjne, ale daje coś, czego nie da się łatwo wycenić: kontrolę nad własnym materiałem poznawczym. A w epoce, w której dane są pamięcią, pamięć jest władzą.

> W świecie, gdzie płynność języka coraz częściej maskuje brak faktów, architektura RAG przywraca należną wagę dokumentowi i źródłu. Ostatecznie jednak żadna baza wektorowa nie stanie się wyrocznią prawdy, a jedynie mapą potencjalnych powiązań. Pytanie brzmi: czy w pogoni za efektywnością retrievalu nie zapomnimy, że między odnalezieniem podobnego sensu a zrozumieniem rzetelnej prawdy wciąż leży przepaść, której nie wypełni żaden algorytm?

Inspiracje

[1]

"Vector Databases A Practical Introduction" Nitin Borwankar

O'Reilly Media | ISBN: 9781098177591

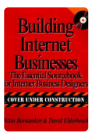
Nitin Borwankar to doświadczony analityk danych i specjalista ds. baz danych z ponad trzydziestoletnim doświadczeniem w rozwoju i wdrażaniu rozwiązań dla przedsiębiorstw. Jest ceniony za wkład w edukację w dziedzinie data science, propagowanie narzędzi open source oraz pracę nad programami nauczania uczenia maszynowego. Borwankar ma wykształcenie inżynierskie i jest związany z sektorem technologicznym od czasu ukończenia IIT Bombay. Jest płodnym autorem i współautorem literatury technicznej, koncentrując się na praktycznych zastosowaniach nowych technologii. Jego najnowsze prace koncentrują się na wektorowych bazach danych, generacji rozszerzonej o wyszukiwanie (RAG) oraz integracji dużych modeli językowych (LLM) z aplikacjami korporacyjnymi. Mieszka w rejonie Zatoki San Francisco i nadal angażuje się w projekty open source oraz pisanie tekstów technicznych.

Nitin Borwankar *is a seasoned data scientist and database professional with over three decades of experience in the development and implementation of enterprise data solutions. He is recognized for his contributions to data science education, his advocacy for open-source tools, and his work on machine learning curricula. Borwankar has a background in engineering and has been involved in the technology sector since graduating from IIT Bombay. He is a prolific author and contributor to technical literature, focusing on practical applications of emerging technologies. His recent work centers on vector databases, retrieval-augmented generation (RAG), and the integration of large language models (LLMs) into enterprise applications. He is based in the San Francisco Bay Area and continues to contribute to open-source projects and technical writing.*

Literatura uzupełniająca

Książki

Autorzy przywoływani w artykule:



[1] **"Building Successful Internet Businesses"**

David Elderbrock, Nitin Borwankar

1996 | Wiley Publishing | ISBN: 9780764570018

Literatura pokrewna (Google Scholar):

[2] **"Building Complex Multi-Agent Systems Using"**

Tim OBrien

[3] **"Natural Language Processing in Action 2E Hobson Lane"**

[4] **"Data Engineering with Generative n Agentic AI"**

Justin J Leto

[5] **"Architecting Intelligent Agents in Azure"**

Hari Narayn

[6] **"Googles PageRank and Beyond Amy N Langville"**

Artykuły naukowe

Autorzy przywoływani:

[1] **"Approximate calculation of the Caputo-type fractional derivative from inaccurate data. Dynamical approach"**

Platon G. Surkov

2021 | Fractional Calculus and Applied Analysis | DOI: 10.1515/fca-2021-0038

Google Scholar

[2] **"Update regarding the interdisciplinary management of the patient under antiresorptive/antiangiogenic/biological therapy in the context of prevention of medication-related osteonecrosis of the jaw"**

Carmen Gabriela Stelea, Alexandra Lorina Platon, Emilia Bologa, Cristina Bologa

2020 | Medic.ro | DOI: 10.26416/med.136.4.2020.3664

Google Scholar

Artykuły pokrewne (Google Scholar):

[3] **"The Laggard's Lock: A Theoretical Model of New Technology Adoption Constrained by Outdated Paradigms"**

Alby Anand Kurian, Nitin Borwankar

2025 | SSRN Electronic Journal | DOI: 10.2139/ssrn.5398026

Google Scholar

Treść tworzy Fundacja Dobre Państwo.
Redaguje i publikuje APA ONE, autorski system redakcyjny Fundacji oparty na AI.
APA ONE Publishing System
Fundacja Dobre Państwo © 2026
Article ID: 44160c16-db8b-432c-8121-ff2e4141f495