



**POLSKO-JAPOŃSKA AKADEMIA
TECHNIK KOMPUTEROWYCH**

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Michał Matejuk

Nr albumu 31624

**Wykorzystanie modeli językowych i algorytmów
dystrybucyjnego osadzania tekstu w analizie treści
internetowych**

Praca magisterska

Promotor: dr inż. Mariusz Trzaska, prof. PJATK

Warszawa, 2026

Streszczenie

Niniejsza praca dotyczy zastosowania modeli językowych oraz algorytmów dystrybucyjnego osadzania tekstu do semantycznej analizy treści internetowych. Opisano w niej projekt, implementację oraz ewaluację modułowego systemu odpowiadania na pytania, opartego na architekturze generowania wspomaganego wyszukiwaniem (ang. *Retrieval-Augmented Generation*, RAG). System pobiera treść artykułów ze wskazanych adresów URL, oczyszcza ją z elementów nawigacyjnych, dzieli na fragmenty i przekształca w wektory semantyczne zapisywane w wektorowej bazie danych. Na etapie zapytania system odnajduje fragmenty najbardziej zbliżone znaczeniowo do pytania użytkownika i na ich podstawie formułuje odpowiedź z odniesieniami do materiału źródłowego.

Część badawcza obejmuje porównanie ośmiu konfiguracji (C1–C8), różniących się modelem językowym, modelem osadzeń oraz strategią podziału tekstu. Ocenie poddano jakość wyszukiwania, merytoryczną jakość odpowiedzi, a także wydajność i koszty eksploatacji. Wyniki wskazują, że dobór modelu językowego ma decydujący wpływ na jakość odpowiedzi, podejście agentowe zapewnia najwyższą wierność kosztem dłuższego czasu i wyższych nakładów finansowych, a niedrogi model chmurowy przewyższa rozwiązanie lokalne pod względem stosunku jakości do kosztu. Pracę zamykają zalecenia wdrożeniowe dotyczące doboru komponentów potoku RAG.

Słowa kluczowe: przetwarzanie języka naturalnego, modele językowe, osadzenia tekstu, generowanie wspomaganie wyszukiwaniem, wyszukiwanie semantyczne, analiza treści internetowych, systemy pytań i odpowiedzi, bazy wektorowe

Spis treści

Streszczenie	i
1 Wstęp	1
1.1 Kontekst i tło problemu	1
1.2 Sformułowanie problemu	1
1.3 Cele badawcze i wkład pracy	2
1.4 Organizacja pracy	3
2 Istniejące rozwiązania — stan sztuki	4
2.1 Wprowadzenie	4
2.2 Ewolucja reprezentacji tekstu	5
2.2.1 <i>Podjęcia tradycyjne</i>	5
2.2.2 <i>Statyczne osadzenia słów</i>	6
2.2.3 <i>Osadzenia kontekstowe</i>	7
2.3 Współczesne modele językowe	8
2.3.1 <i>BERT i modele typu enkoder</i>	8
2.3.2 <i>Zamknięte duże modele językowe</i>	8
2.3.3 <i>Otwarte duże modele językowe</i>	9
2.3.4 <i>Modele multimodalne</i>	10
2.4 Osadzenia zdań i dokumentów	10
2.4.1 <i>Od osadzeń słów do osadzeń zdań</i>	11
2.4.2 <i>Współczesne modele osadzeń</i>	11
2.4.3 <i>Benchmarki ewaluacyjne</i>	12
2.5 Bazy wektorowe i wyszukiwanie podobieństwa	12
2.5.1 <i>Potrzeba baz wektorowych</i>	12
2.5.2 <i>Popularne rozwiązania baz wektorowych</i>	13
2.5.3 <i>Algorytmy indeksowania</i>	13
2.5.4 <i>Metryki odległości</i>	14
2.6 Generowanie wspomagane wyszukiwaniem (RAG)	15
2.6.1 <i>Motywacja i koncepcja</i>	15
2.6.2 <i>Architektura potoku RAG</i>	16
2.6.3 <i>Strategie dzielenia tekstu na fragmenty</i>	16
2.6.4 <i>Zaawansowane techniki RAG</i>	17
2.6.5 <i>Ewaluacja systemów RAG</i>	18
2.7 Zastosowania w analizie treści internetowych	19
2.7.1 <i>Ekstrakcja treści ze stron internetowych</i>	19
2.7.2 <i>Zadania analizy semantycznej</i>	19
2.7.3 <i>Odpowiadanie na pytania dotyczące treści internetowych</i>	20
2.7.4 <i>Przewagi nad wyszukiwaniem słów kluczowych</i>	20
2.8 Podsumowanie rozdziału	21
3 Założenia i metodologia proponowanego rozwiązania	22

3.1	Podjęcie badawcze	22
3.1.1	<i>Pytania badawcze</i>	22
3.1.2	<i>Ramy metodologiczne</i>	23
3.2	Projekt architektury systemu	23
3.2.1	<i>Ogólny projekt systemu</i>	23
3.2.2	<i>Potok indeksowania treści</i>	24
3.2.3	<i>Potok przetwarzania zapytań</i>	25
3.2.4	<i>Diagram przepływu danych</i>	26
3.3	Metodologia przetwarzania danych	26
3.3.1	<i>Charakterystyka zbioru testowego</i>	26
3.3.2	<i>Strategie dzielenia tekstu na fragmenty</i>	27
3.3.3	<i>Ekstrakcja metadanych</i>	27
3.3.4	<i>Potok przetwarzania wstępnego</i>	27
3.4	Metodologia ewaluacji	28
3.4.1	<i>Wymiary ewaluacji</i>	28
3.4.2	<i>Metryki wyszukiwania</i>	28
3.4.3	<i>Metryki generowania odpowiedzi</i>	29
3.4.4	<i>Metryki wydajności</i>	30
3.4.5	<i>Plan eksperymentów</i>	30
3.4.6	<i>Tworzenie danych referencyjnych</i>	31
3.4.7	<i>Analiza statystyczna</i>	31
3.5	Kwestie etyczne	31
3.5.1	<i>Prywatność danych i ochrona informacji</i>	31
3.5.2	<i>Prawa autorskie i dozwolony użytek</i>	31
3.5.3	<i>Stronniczość modeli i wiarygodność informacji</i>	32
3.5.4	<i>Wpływ środowiskowy</i>	32
3.6	Ograniczenia aplikacyjne	32
3.6.1	<i>Zakres rozwiązania</i>	32
3.6.2	<i>Ograniczenia techniczne i jakościowe</i>	32
3.7	Podsumowanie rozdziału	33
4	Zastosowane technologie i narzędzia	34
4.1	Język programowania — Python	34
4.2	Framework RAG — Haystack	34
4.3	Framework webowy — FastAPI	34
4.4	Wybór modelu osadzeń	35
4.5	Baza wektorowa — ChromaDB	35
4.6	Wybór modelu językowego	36
4.7	Biblioteki do ekstrakcji treści ze stron	36
4.8	Dodatkowe biblioteki	36
5	Implementacja systemu	38
5.1	Wprowadzenie i stos technologiczny	38
5.2	Architektura systemu i przepływ danych	38
5.3	Implementacja głównych komponentów	40
5.3.1	<i>Moduł konfiguracji systemu i zarządzania ustawieniami</i>	40
5.3.2	<i>Fabryka generatorów i obsługa interfejsów API</i>	41
5.3.3	<i>Rozwiązanie problemu oczyszczania metadanych (MetadataCleaner)</i>	41
5.4	Implementacja potoków przetwarzania (Pipelines)	43
5.4.1	<i>Potok indeksowania</i>	43

5.4.2	Potok zapytań	44
5.5	Warstwa usług oraz interfejs API	44
5.6	Wdrożenie i konteneryzacja	45
5.7	Podsumowanie rozdziału	46
6	Testy i wyniki	47
6.1	Wprowadzenie	47
6.1.1	Cele testowania	47
6.1.2	Pytania badawcze	47
6.2	Metodologia testów	48
6.2.1	Plan eksperymentów	48
6.2.2	Konfiguracje testowe	48
6.2.3	Zbiór danych testowych	48
6.2.4	Metryki	51
6.2.5	Procedura testowa	52
6.2.6	Hipotezy	52
6.3	Konfiguracja eksperymentów	53
6.3.1	Szczegóły implementacyjne	53
6.3.2	Proces zbierania danych	54
6.4	Wyniki	54
6.4.1	Wyniki ilościowe	54
6.4.2	Wyniki jakościowe	58
6.5	Analiza i dyskusja	60
6.5.1	Weryfikacja hipotez	60
6.5.2	Analiza kompromisów (<i>Trade-off Analysis</i>)	62
6.5.3	Ograniczenia i zagrożenia dla trafności (<i>Threats to Validity</i>)	64
6.6	Podsumowanie rozdziału	64
6.6.1	Odpowiedzi na pytania badawcze	64
6.6.2	Rekomendacje praktyczne	65
6.6.3	Kierunki dalszych badań	66
7	Podsumowanie i wnioski końcowe	67
7.1	Realizacja celów pracy	67
7.2	Wady i zalety zaproponowanego rozwiązania	67
7.2.1	Zalety systemu	67
7.2.2	Wady i ograniczenia systemu	68
7.3	Kierunki dalszych badań i plany na przyszłość	68
7.4	Wnioski końcowe	69
	Bibliografia	70
A	Słownik użytej terminologii i skrótów	74
	Spis rysunków	77
	Spis tabel	78
	Spis listingów	79

Rozdział 1

Wstęp

Przedmiotem niniejszej pracy jest zastosowanie modeli językowych oraz algorytmów dystrybucyjnego osadzania tekstu do semantycznej analizy treści internetowych. Proces ten pozwala na automatyczną ekstrakcję wiedzy. W niniejszym rozdziale nakreślono tło metodologiczne, sformułowano problem badawczy oraz zdefiniowano cele pracy. Opisano tu również wkład naukowy i przedstawiono strukturę dokumentu.

1.1 Kontekst i tło problemu

Masowy przyrost publikacji w sieci wymusza ciągle doskonalenie metod filtracji i selekcji danych. Codziennie powstają miliony artykułów, wpisów blogowych czy komunikatów prasowych. Z perspektywy odbiorcy zasadnicza staje się zdolność do szybkiego odnajdywania precyzyjnych odpowiedzi. Klasyczne metody selekcji zawodzą.

Tradycyjne systemy wyszukiwania opierają się na indeksowaniu słów kluczowych. Rozwiązanie to jest wydajne, lecz obciążone istotną wadą. Ignoruje ono bowiem semantykę zapytania oraz samych dokumentów. Przykładowo, zapytanie o „wpływ zmian klimatycznych na rolnictwo” może pominąć wartościowe teksty omawiające „skutki globalnego ocieplenia dla upraw” lub „wpływ rosnących temperatur na plony”. Maszyna bez modeli semantycznych nie powiąże tych pojęć.

Rozwój technik przetwarzania języka naturalnego (ang. *Natural Language Processing*, NLP), zwłaszcza metod uczenia głębokiego, zmienił ten stan rzeczy. Zaproponowanie architektury Transformer przez Vaswaniego i współpracowników [1] zredefiniowało kierunek badań. Maszyny zyskały zdolność analizy kontekstowej. Modele takie jak BERT [2] wykazały, że wstępnie wytrenowane reprezentacje językowe dobrze odzwierciedlają relacje semantyczne między słowami i frazami.

Kolejnym etapem ewolucji stały się duże modele językowe (ang. *Large Language Models*, LLMs). Wykazują one wysoką sprawność w analizie kontekstu i generowaniu spójnych wypowiedzi. Reprezentują je systemy takie jak GPT-5.5 [3], Claude Sonnet 4.6 [4], Gemini 2.5 Pro [5], LLaMA 4 [6] oraz Grok 4 [7]. Każdy z nich ma unikalne cechy użytkowe. Ich bezpośrednie użycie bywa jednak utrudnione ze względu na ograniczenia w dostępie do aktualnej wiedzy. Problem ten rozwiązuje architektura generowania wspomaganego wyszukiwaniem (ang. *Retrieval-Augmented Generation*, RAG), sformułowana przez Lewisa i współpracowników [8]. RAG łączy generatywne zdolności LLM z precyzją zewnętrznych baz danych. Umożliwia to weryfikację faktów.

1.2 Sformułowanie problemu

Mimo rozwoju wyszukiwarek ekstrakcja konkretnych informacji z artykułów internetowych wciąż napotyka na poważne przeszkody. Główne problemy obejmują:

1. **Przeładowanie informacyjne:** Nadmiar publikowanych treści utrudnia selekcję wiarygodnych źródeł. Odnalezienie odpowiedzi na proste pytania wymaga nierzadko lektury wielu obszernych tekstów.
2. **Optymalizacja pod kątem wyszukiwarek (SEO):** Wiele artykułów projektuje się z myślą o algorytmach pozycjonujących, a nie o wartości merytorycznej. Nagłówki typu *clickbait* oraz teksty sztucznie nasyczone frazami kluczowymi obniżają użyteczność sieci.

3. **Niedopasowanie semantyczne:** Algorytmy leksykalne nie radzą sobie z synonimami i różnicami słownikowymi. Sposób formułowania pytań przez użytkowników rzadko pokrywa się dokładnie z zapisem autorów dokumentu.
4. **Zależność od kontekstu:** Prawidłowa interpretacja pojedynczego twierdzenia w tekście wymaga zazwyczaj znajomości całego artykułu. Spowalnia to proces wyszukiwania drobnych faktów.
5. **Brak bezpośrednich odpowiedzi:** Klasyczne wyszukiwarki zwracają jedynie listę odnośników. Cała praca związana z analizą dokumentu i ekstrakcją wniosków spoczywa na odbiorcy.

Rozwiązanie to wymaga zmiany paradygmatu. Należy przejść od prostego indeksowania słów do semantycznej interpretacji całych tekstów. Modele językowe umożliwiają precyzyjną ekstrakcję wiedzy bezpośrednio z kontekstu źródłowego.

1.3 Cele badawcze i wkład pracy

W pracy analizuje się możliwości modeli językowych oraz algorytmów dystrybucyjnego osadzania tekstu w przetwarzaniu witryn sieciowych. Cel główny stanowi zaprojektowanie, implementacja i ewaluacja modułowego systemu odpowiadania na pytania. System ten powinien umożliwiać:

1. Pobieranie i parsowanie treści stron internetowych na podstawie podanego adresu URL bądź wyszukiwanie ich w lokalnym repozytorium.
2. Zadawanie zapytań w języku naturalnym odnośnie do zawartości zgromadzonych materiałów.
3. Generowanie zwięzłych i wiarygodnych odpowiedzi osadzonych w tekście źródłowym wraz ze wskazaniem odpowiednich akapitów.

Realizacja celu wymaga odpowiedzi na następujące pytania badawcze:

- **RQ1:** W jakim stopniu wybrane techniki osadzania tekstu różnią się pod względem precyzji odwzorowania treści internetowych?
- **RQ2:** Jaka konfiguracja parametrów systemu RAG zapewnia najwyższą jakość odpowiedzi w zadaniach analizy serwisów informacyjnych?
- **RQ3:** Jak podział tekstu na fragmenty, wybór modelu osadzeń i algorytm wyszukiwania wpływają na dokładność generowanych odpowiedzi?
- **RQ4:** Jakie kompromisy eksploatacyjne (opóźnienia, koszty finansowe, zapotrzebowanie na zasoby) wiążą się z wdrożeniem takiego systemu w warunkach produkcyjnych?

Wkład naukowy i praktyczny pracy obejmuje:

- Systematyczny przegląd literatury przedmiotu w zakresie modeli osadzeń, dużych modeli językowych oraz architektur RAG.
- Projekt modułowej architektury systemu analitycznego integrującej bazy wektorowe oraz otwarte interfejsy programistyczne modeli językowych.
- Badania empiryczne porównujące efektywność modeli językowych i parametryzację potoków przetwarzania tekstu.
- Opracowanie zaleceń wdrożeniowych dotyczących implementacji systemów typu QA opartych na technologii RAG.

1.4 Organizacja pracy

Struktura pracy odzwierciedla kolejne etapy prowadzonych badań oraz prac implementacyjnych.

Rozdział drugi (*Stan sztuki*) przybliża ewolucję reprezentacji tekstu. Przedstawiono w nim rozwój algorytmów transformacyjnych, baz wektorowych i systemów wspomaganych wyszukiwaniem.

W rozdziale trzecim (*Założenia i metodologia*) opisano ramy badawcze. To tutaj zdefiniowano architekturę przepływu danych, scenariusze testowe oraz hipotezy badawcze.

Rozdział czwarty (*Zastosowane technologie i narzędzia*) omawia język programowania, framework webowy, bazę wektorową oraz modele językowe i modele osadzeń wykorzystane w prototypie.

Rozdział piąty (*Implementacja*) dokumentuje techniczną realizację systemu. Opisano w nim strukturę kodu, budowę potoków przetwarzania oraz integrację poszczególnych modułów.

Rozdział szósty (*Testy i wyniki*) prezentuje rezultaty eksperymentów empirycznych. Obejmują one statystyczną analizę danych ilościowych oraz ocenę jakościową generowanych odpowiedzi.

Prace zamyka rozdział siódmy (*Podsumowanie*), który stanowi syntezę uzyskanych wyników. Wskazano w nim napotkane ograniczenia techniczne oraz zarysowano kierunki dalszego rozwoju systemu. Słownik pojęć i skrótów umieszczono w Dodatku A.

Rozdział 2

Istniejące rozwiązania — stan sztuki

Niniejszy rozdział poświęcono analizie stanu wiedzy w obszarze reprezentacji tekstu, modeli językowych, baz wektorowych oraz systemów generowania wspomaganego wyszukiwaniem (*Retrieval-Augmented Generation* – RAG). Zrozumienie tych pojęć stanowi warunek konieczny do zaprojektowania i implementacji systemu analizy treści sieciowych. W pierwszej kolejności zdefiniowano tu podstawową terminologię. Prześladowano też ewolucję metod kodowania znaczenia i strukturę współczesnych modeli językowych. Osobne sekcje opisują techniki osadzania zdań, mechanizmy indeksowania baz wektorowych oraz architekturę rozwiązań typu RAG. Całość zamyka omówienie praktycznych zastosowań tych technologii w analityce portali internetowych.

2.1 Wprowadzenie

Przetwarzanie języka naturalnego (ang. *Natural Language Processing* – NLP) to fundament współczesnej sztucznej inteligencji. Główny cel tej dziedziny sprowadza się do umożliwienia maszynom interpretacji, analizy oraz generowania wypowiedzi ludzkich w sposób zbliżony do naturalnego. Początkowo analizę lingwistyczną opierano na tokenizacji, czyli podziale tekstu na mniejsze jednostki słownikowe. Powszechnie stosowano lematyzację oraz stemming. Służyły one odpowiednio do sprowadzania wyrazów do form podstawowych i heurystycznego usuwania końcówek fleksyjnych. Strukturyzację pozyskiwanych korpusów tekstowych realizowano ponadto poprzez znakowanie części mowy (ang. *Part-of-Speech Tagging* – POS Tagging) i rozpoznawanie nazw własnych (ang. *Named Entity Recognition* – NER).

Największą trudność w NLP stanowiło numeryczne ujęcie relacji semantycznych. Chodzi o precyzyjne określenie bliskości znaczeniowej dwóch fragmentów tekstu bez względu na ich formę zapisu. Przełom przyniosło modelowanie przestrzeni wektorowej. W podejściu tym słowa bądź całe zdania przyjmują postać gęstych wektorów. Te wielowymiarowe struktury numeryczne charakteryzują się brakiem wartości zerowych na większości pozycji. Koncepcja ta wynika bezpośrednio z hipotezy dystrybucyjnej J.R. Firtha [9]. Zgodnie z nią znaczenie wyrazu definiuje kontekst jego występowania, czyli słowa w bezpośrednim sąsiedztwie. W konsekwencji bliskość geometryczna wektorów w przestrzeni (zazwyczaj o wymiarowości od 384 do 4096 cech) odzwierciedla pokrewieństwo pojęciowe. Do obliczania tej odległości stosuje się najczęściej podobieństwo cosinusowe lub dystans euklidesowy.

Obecnie w dziedzinie NLP dominują duże modele językowe (ang. *Large Language Models* – LLMs). Konstruuje się je jako wielowarstwowe sieci neuronowe oparte na architekturze Transformer [1]. Centralną rolę odgrywa tu mechanizm uwagi (ang. *Attention Mechanism*), ze szczególnym uwzględnieniem samo-uwagi (ang. *Self-Attention*). Umożliwia on dynamiczne określanie wag dla poszczególnych elementów sekwencji wejściowej podczas przetwarzania każdego tokenu. Rozwiązuje to problem ograniczeń starszych architektur rekurencyjnych (jak LSTM), pozwalając na jednoczesną analizę odległych zależności. Trening modeli przebiega dwuetapowo. W fazie wstępnej (ang. *Pre-training*) sieć przyswaja ogólne reguły gramatyczne i semantyczne na podstawie ogromnych, nieoznaczonych korpusów tekstowych. Następnie realizuje się dostrajanie (ang. *Fine-tuning*) nakierowane na konkretne zadania dziedzinowe. Większość LLMs funkcjonuje autoregresyjnie. Oznacza to generowanie tekstu słowo po słowie, gdzie wybór kolejnego elementu zależy od całej dotychczasowej sekwencji. Zakres tej zależności ogranicza jednak rozmiar okna kontekstowego, czyli maksymalna liczba tokenów przetwarzanych jednorazowo przez sieć.

Istniejącą słabością LLMs pozostaje podatność na halucynacje. Przejawia się ona generowaniem

treści logicznie spójnych, lecz niezgodnych z rzeczywistością lub niepopartych źródłami. Wynika to z zamrożenia wiedzy parametrycznej modelu w jego wagach po zakończeniu treningu. Do rozwiązania tego problemu służy generowanie wspomagane wyszukiwaniem (ang. *Retrieval-Augmented Generation* – RAG). Metoda ta łączy mechanizmy pobierania wiedzy z zewnętrznych źródeł z procesem syntezy tekstu. Klasyczne wyszukiwanie rzadkie (ang. *Sparse Retrieval*), opierające się na dopasowywaniu słów kluczowych (np. BM25), ustępuje tu miejsca metodom gęstym (ang. *Dense Retrieval*). Wykorzystują one wielowymiarowe osadzenia semantyczne do identyfikacji najbardziej relewantnych fragmentów dokumentacji.

Uruchomienie systemu RAG wymaga dedykowanej infrastruktury. Do przechowywania i indeksowania wektorów stosuje się wyspecjalizowane bazy wektorowe. Zamiast kosztownego wyszukiwania dokładnego, bazy te realizują algorytmy przybliżonych najbliższych sąsiadów (ang. *Approximate Nearest Neighbor* – ANN). Wymaga to uprzedniego podzielenia tekstów na mniejsze fragmenty (ang. *chunking*) i wygenerowania dla nich wektorów. W fazie zapytania przydatne okazuje się także ponowne ocenianie (ang. *reranking*), które porządkuje wyniki przed przekazaniem ich do generatora. Integracja tych technologii pozwala na budowę systemów zdolnych do precyzyjnego odpowiadania na pytania w oparciu o dynamicznie zmieniające się zasoby internetowe, co stanowi główny cel badawczy i aplikacyjny niniejszej pracy.

2.2 Ewolucja reprezentacji tekstu

Przechowywanie i interpretacja tekstu przez komputery przeszły długą drogę. Przejście od reprezentacji symbolicznych do gęstych wektorów znacząco poprawiło maszynowe rozumienie języka. Analizie poddano metody rzadkie, statyczne oraz nowoczesne osadzenia kontekstowe.

2.2.1 Podejścia tradycyjne

Początkowo stosowano podejście torby słów (ang. *Bag-of-Words* – BoW). W modelu tym tekst stanowi nieuporządkowany zbiór wyrazów. Gramatyka, składnia oraz kolejność występowania słów są całkowicie pomijane. Dokument jest rzutowany na wektor o wymiarowości równej rozmiarowi całego słownika korpusu (V). Poszczególne współrzędne odpowiadają częstościom występowania danych słów. Rozwiązanie to cechuje prostota i szybkość obliczeniowa. Ignoruje ono jednak relacje semantyczne. Wyrazy o zbliżonym znaczeniu, jak „samochód” i „auto”, są traktowane jako wymiary ortogonalne. Uniemożliwia to automatyczne wykrywanie podobieństwa pojęciowego.

Pewne udoskonalenie stanowi metoda ważenia TF-IDF (*Term Frequency-Inverse Document Frequency*) [10]. Zasada jej działania opiera się na prostym założeniu: słowa kluczowe powinny dominować w pojedynczym dokumencie, lecz rzadko występować w skali całego korpusu. Ułatwia to odrzucanie wyrazów o charakterze czysto syntaktycznym. Obliczenie wartości TF-IDF dla termu t w dokumencie d opisuje wzór (2.1):

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log \frac{N}{\text{DF}(t)} \quad (2.1)$$

Zmienna $\text{TF}(t, d)$ odpowiada częstości termu t w dokumencie d . Przez N oznaczono łączną liczbę dokumentów, a $\text{DF}(t)$ wskazuje, w ilu z nich dany term wystąpił co najmniej raz.

Kontekst lokalny pozwalają uchwycić n -gramy. Tworzy się je jako sekwencje składające się z n kolejnych słów lub znaków. Zastosowanie bigramów ($n = 2$) lub trigramów ($n = 3$) umożliwia detekcję prostych związków frazeologicznych. Niestety podejście to powoduje wykładniczy wzrost wymiarowości przestrzeni cech. Konsekwencją jest skrajna rzadkość danych (ang. *data sparsity*).

Główne ograniczenia metod symbolicznych obejmują:

- Pomijanie podobieństwa semantycznego, w wyniku czego wyrazy bliskoznaczne otrzymują wzajemnie ortogonalne wektory.
- Wrażliwość na niedopasowanie słownictwa (ang. *vocabulary mismatch*) między zapytaniem a przeszukiwanym tekstem.
- Ogromną wymiarowość oraz rzadkość wektorów, co utrudnia przetwarzanie dużych zbiorów danych.
- Ignorowanie szyku zdania oraz relacji składniowych poza zakresem krótkiego n-gramu.

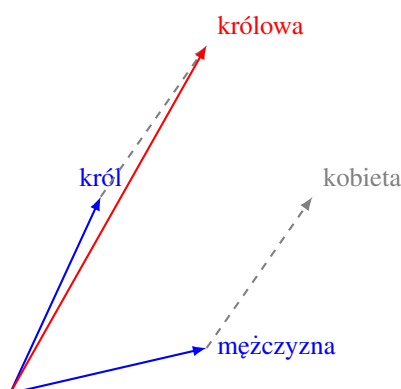
2.2.2 Statyczne osadzenia słów

Kolejny przełom nastąpił wraz z wprowadzeniem statycznych osadzeń słów (ang. *word embeddings*). W tym ujęciu słowo staje się gęstym wektorem w ciągłej przestrzeni o stałej, stosunkowo niskiej wymiarowości. Standardowo stosuje się od 100 do 300 wymiarów. Podobieństwo semantyczne przekłada się tu bezpośrednio na geometryczną bliskość wektorów.

Szerokie zastosowanie zyskała rodzina modeli Word2Vec [11]. Mikołov zaproponował dwa warianty architektoniczne płytkich sieci neuronowych z jedną warstwą ukrytą:

- **Skip-gram**: przewiduje otoczenie (słowa kontekstowe) na podstawie wybranego wyrazu docelowego. Podejście to cechuje wysoka skuteczność w reprezentowaniu rzadkich pojęć.
- **CBOW** (ang. *Continuous Bag-of-Words*): wyznacza słowo docelowe na podstawie otaczającego go kontekstu. Wariant ten charakteryzuje się krótszym czasem uczenia i lepszymi wynikami dla wyrazów powtarzających się często.

Zaletą Word2Vec okazała się liniowość wyuczonej przestrzeni. Umożliwia to wykonywanie operacji arytmetycznych bezpośrednio na wektorach. Klasycznym przykładem jest relacja: $\vec{v}_{\text{król}} - \vec{v}_{\text{mężczyzna}} + \vec{v}_{\text{kobieta}} \approx \vec{v}_{\text{królowa}}$. Potwierdza to zdolność modelu do kodowania abstrakcyjnych cech semantycznych, takich jak płeć lub czas gramatyczny. Rysunek 2.1 ilustruje tę zależność w uproszczonej, dwuwymiarowej przestrzeni wektorowej.



Rysunek 2.1: Uproszczona ilustracja liniowości przestrzeni wektorowej Word2Vec: dodanie różnicy wektorów „kobieta” i „mężczyzna” do wektora „król” daje wektor bliski wektorowi „królowa”.

Alternatywną metodę stanowi model GloVe (*Global Vectors for Word Representation*) [12]. Zamiast ograniczać się do lokalnych okien kontekstowych, łączy on zalety globalnej faktoryzacji macierzy z metodami predykcyjnymi. Proces opiera się na budowie globalnej macierzy współwystępowania wyrazów w całym korpusie. Model minimalizuje funkcję kosztu bazującą na logarytmie prawdopodobieństwa wspólnego występowania słów.

Rozwinięcie tej koncepcji przyniósł model FastText [13]. Wprowadził on podział słów na podsłowa (n-gramy znakowe). Wektor wyrazu powstaje tu jako suma wektorów jego składowych. Podejście to eliminuje problem słów spoza słownika treningowego (ang. *out-of-vocabulary* – OOV). Ułatwia również analizę języków o bogatej fleksji i morfologii, takich jak język polski.

Metody statyczne obarczone są jednak poważną wadą. Przypisują one wyrazom niezmienną reprezentację wejściową niezależnie od kontekstu. Wyklucza to poprawną interpretację homonimów. Przykładowo, słowo „zamek” otrzyma identyczne osadzenie w zdaniu o twierdzy obronnej, mechanizmie w drzwiach oraz suwaku odzieży. Ponadto modele te nie potrafią bezpośrednio reprezentować całych zdań ani dokumentów bez uproszczonych metod agregacji, takich jak uśrednianie wektorów składowych.

2.2.3 Osadzenia kontekstowe

Problem ten rozwiązały osadzenia kontekstowe. W tym ujęciu reprezentacja wektorowa powstaje dynamicznie na podstawie otoczenia słowa. Jednym z pierwszych rozwiązań tego typu był model ELMo (*Embeddings from Language Models*) [14]. Wykorzystuje on dwukierunkowe sieci LSTM do łączenia stanów ukrytych z przebiegu w obu kierunkach. Mimo poprawy wyników w zadaniach NLP, sekwencyjna natura LSTM utrudniała równoległe obliczenia i modelowanie odległych zależności.

Rozwiązanie przyniosła architektura Transformer [1]. Zastąpiła ona rekurencję mechanizmem samouwagi (ang. *self-attention*). Umożliwia to równoległą analizę całego tekstu przy jednoczesnym wyznaczaniu wag powiązań semantycznych i syntaktycznych. Model opiera się na trzech macierzach: zapytań (Q), kluczy (K) oraz wartości (V). Obliczenie uwagi przebiega zgodnie ze wzorem (2.2):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.2)$$

Parametr d_k określa wymiar wektorów kluczy, pełniąc rolę czynnika skalującego. Zapobiega on zaniżaniu gradientu podczas obliczania funkcji softmax. Dzięki uwadze wielogłowicowej (ang. *Multi-Head Attention*) proces ten przebiega równoległe w kilku podprzestrzeniach. Pozwala to modelowi na jednoczesną analizę relacji o różnym charakterze syntaktycznym i semantycznym.

Podsumowanie i porównanie omówionych metod reprezentacji tekstu przedstawiono w Tabeli 2.1.

Tabela 2.1: Porównanie głównych metod reprezentacji tekstu

Metoda	Typ reprezentacji	Główne zalety	Wymiarowość
Bag-of-Words	Rzadka, oparta na częstotliwościach	Prostota obliczeniowa, łatwość interpretacji	Rozmiar słownika (V)
TF-IDF	Rzadka, ważona znaczeniem słów	Skuteczne odrzucanie szumu, prosta implementacja	Rozmiar słownika (V)
Word2Vec / GloVe	Gęsta, statyczna semantycznie	Przechwytywanie synonimii i relacji semantycznych	Gęsta (100 – 300)
FastText	Gęsta, statyczna podsłownowa	Obsługa słów spoza słownika, analiza morfologiczna	Gęsta (100 – 300)
BERT / Transformer	Gęsta, dynamiczna (kontekstowa)	Rozwiązywanie problemu wieloznaczności, pełny kontekst	Gęsta (384 – 4096)

2.3 Współczesne modele językowe

Kolejne generacje modeli opartych na architekturze Transformer poprawiają wyniki w standardowych testach rozumienia i generowania języka, takich jak MMLU [15] czy SuperGLUE [16], a w części zadań osiągają wyniki bliskie poziomowi eksperckiemu. Postęp ten wynika z pre-trainingu na potężnych wolumenach danych oraz późniejszej adaptacji dziedzinowej. W tym miejscu scharakteryzowano architekturę BERT. Omówiono też wiodące zamknięte i otwarte modele generatywne oraz systemy multimodalne przydatne w analizie zawartości stron internetowych.

2.3.1 BERT i modele typu enkoder

Dużą zmianę w rozumieniu tekstu przyniosła rodzina modeli BERT (*Bidirectional Encoder Representations from Transformers*) [2]. W przeciwieństwie do modeli jednokierunkowych, BERT analizuje kontekst dwustronnie w każdej warstwie sieci. Umożliwia to lepsze odwzorowanie zależności międzywyrazowych. Uczenie wstępne tego modelu opiera się na dwóch zadaniach:

- **Maskowane modelowanie języka** (ang. *Masked Language Modeling* – MLM): polegające na maskowaniu 15% tokenów wejściowych w celu ich późniejszego odtworzenia z kontekstu dwukierunkowego.
- **Przewidywanie kolejnego zdania** (ang. *Next Sentence Prediction* – NSP): klasyfikacja binarna sprawdzająca, czy dane zdanie naturalnie następuje po poprzednim w oryginalnym korpusie.

BERT zadebiutował w dwóch konfiguracjach:

- **BERT-BASE**: wersja o 12 warstwach enkodera, posiadająca 110 milionów parametrów oraz wektory ukryte o rozmiarze 768.
- **BERT-LARGE**: wersja o 24 warstwach enkodera, posiadająca 340 milionów parametrów oraz wektory ukryte o rozmiarze 1024.

Popularność tego rozwiązania zrodziła liczne modyfikacje. Do najważniejszych odmian należą:

- **RoBERTa** [17]: wariant pozbawiony zadania NSP, uczony na znacznie większych wolumenach danych przy użyciu dynamicznego maskowania tokenów.
- **DistilBERT** [18]: model odchudzony metodą destylacji wiedzy (ang. *knowledge distillation*), w której mniejsza sieć naśladuje zachowanie większego odpowiednika. Jest o 40% mniejszy i o 60% szybszy od bazy, zachowując niemal całą jej skuteczność.
- **ALBERT** [19]: redukuje liczbę parametrów poprzez faktoryzację parametrów osadzeń oraz współdzielenie wag, ułatwiając trening głębszych modeli.

Słabością enkoderów pozostaje jednak nieefektywność w generowaniu dłuższego tekstu. Przeznacza się je zatem do klasyfikacji, ekstrakcji informacji oraz generowania wektorów semantycznych.

2.3.2 Zamknięte duże modele językowe

Rozwój LLMs doprowadził do powstania komercyjnych systemów API o wysokich zdolnościach wnioskowania. Do wiodących rozwiązań zamkniętych należą:

- **GPT-5.5** [3]: zunifikowany, multimodalny model typu *omni* od OpenAI, przetwarzający w ramach jednej sesji tekst, kod, dźwięk, obrazy i wideo. Oferuje okno kontekstowe o rozmiarze 128 000 tokenów. Bezpośrednia integracja modalności eliminuje opóźnienia i narzut systemów kaskadowych. Model cechuje wysoka jakość syntezy w języku polskim oraz zaawansowane wnioskowanie wieloetapowe.

- **Claude Sonnet 4.6** [4]: flagowy model firmy Anthropic, zoptymalizowany pod kątem zadań programistycznych oraz operacji agentywnych. Dysponuje oknem kontekstowym rzędu 1 miliona tokenów. Dzięki koncepcji wnioskowania hybrydowego (ang. *hybrid reasoning*) model potrafi realizować wieloetapowe, autonomiczne zadania. Integruje się z systemem operacyjnym przy użyciu mechanizmów kontroli komputera (ang. *computer control*).
- **Claude 4.1 Opus** [20]: wariant wyspecjalizowany Anthropic, przeznaczony do wieloetapowej analizy w trudnych domenach naukowych, prawnych i finansowych. Posiada wewnętrzną przestrzeń myślową (ang. *Thinking Space*) o rozmiarze 64 000 tokenów, służącą do planowania odpowiedzi.
- **Gemini 2.5 Pro** [5]: silnik opracowany przez Google DeepMind, wyróżniający się oknem kontekstowym sięgającym od 1 do 2 milionów tokenów. Zapewnia natywną wielomodalność, w tym bezpośrednią generację i interpretację mowy w ponad 24 językach z zachowaniem intonacji.
- **Grok 4** [7]: model od xAI, połączony bezpośrednio z serwisem X w celu analizy informacji bieżących w czasie rzeczywistym. Jego okno kontekstowe mieści 256 000 tokenów. Konstrukcja kładzie nacisk na sprawne rozwiązywanie zadań matematycznych i programistycznych.

Wybrane parametry oraz cechy charakterystyczne wiodących dużych modeli językowych zestawiono w Tabeli 2.2.

Tabela 2.2: Porównanie wiodących dużych modeli językowych (stan na koniec 2025 r.)

Model	Dostawca	Dostępność	Okno kontekstu	Główny wyróżnik
GPT-5.5	OpenAI	Zamknięty	128 000 tokenów	Zaawansowane wnioskowanie wieloetapowe, głębokie planowanie logiczne
Claude Sonnet 4.6	Anthropic	Zamknięty	200 000 – 1 000 000	Najlepsze wyniki w zadaniach programistycznych i agentywnych
Claude 4.1 Opus	Anthropic	Zamknięty	200 000 tokenów	Zaawansowane wnioskowanie logiczne, finanse i prawo
Gemini 2.5 Pro	Google	Zamknięty	1 000 000 – 2 000 000	Największe okno kontekstu, natywna wielomodalność (audio)
Grok 4	xAI	Zamknięty	256 000 tokenów	Dostęp w czasie rzeczywistym do platformy społecznościowej X
LLaMA 4 (Meta)	Meta	Otwarty	do 10 000 000 tokenów	Pierwszy otwartoźródłowy model MoE, natywna wielomodalność
Mixtral 8x7B	Mistral AI	Otwarty	32 000 tokenów	Architektura <i>Mixture of Experts</i> , wysoka wydajność lokalna

2.3.3 Otwarte duże modele językowe

Szerokie zastosowanie zyskują rozwiązania o otwartych wagach (ang. *open-weights*). Umożliwiają one pełną prywatność, lokalne hostowanie i niezależność kosztową od zewnętrznych interfejsów API. Grupę tę reprezentują następujące systemy:

- **LLaMA 4** [6]: najnowsza rodzina modeli od Meta (wersje Scout oraz Maverick). Wykorzystuje architekturę *Mixture of Experts* (MoE) i natywną wielomodalność. Udostępnia okno kontekstowe mieszczące do 10 milionów tokenów.
- **LLaMA 3** [21]: rodzina modeli wydana w wariantach 8B, 70B i 405B. Wersja 405B z powodzeniem rywalizuje z zamkniętymi komercyjnymi odpowiednikami. Otwarta licencja ułatwiła społeczności stworzenie tysięcy wyspecjalizowanych wariantów dziedziny.
- **Modele Mistral** [22]: rozwijane przez firmę Mistral AI. Warianty te spopularyzowały architekturę uwagi z zapytaniami grupowymi (ang. *Grouped-Query Attention* – GQA) oraz mechanizmy MoE (Mixtral 8x7B), co pozwala na osiąganie dobrych rezultatów przy niskich wymaganiach sprzętowych.
- **Qwen 3** [23]: seria opracowana przez grupę Alibaba, ceniona za sprawne tłumaczenia oraz wielojęzyczność, w tym wysoką precyzję w zadaniach matematycznych.
- **DeepSeek V3** [24]: chiński model o architekturze MoE, wykazujący zbliżone możliwości do komercyjnych liderów rynku przy niskich kosztach uczenia i eksploatacji.
- **Phi-3 / Phi-4** [25]: reprezentanci klasy małych modeli językowych (ang. *Small Language Models* – SLMs). Optymalizuje się je pod kątem działania bezpośrednio na urządzeniach końcowych. Ich atutem jest trening na starannie dobranych danych wysokiej jakości.

Decyzja o wyborze typu modelu wymaga analizy kompromisów. Zamknięte rozwiązania API przodują w skomplikowanym wnioskowaniu logicznym. Są jednak zależne od łączności sieciowej i polityki prywatności dostawców. Wdrożenia otwartoźródłowe gwarantują natomiast niezależność i stabilność kosztową kosztem wyższych wymagań dotyczących własnej infrastruktury obliczeniowej.

2.3.4 Modele multimodalne

Ważny kierunek rozwoju stanowi natywna wielomodalność. Zapewnia ona obsługę obrazu, dźwięku, tekstu oraz wideo. Początek tym rozwiązaniom dał model CLIP (*Contrastive Language-Image Pre-training*) [26]. Dzięki nauce kontrastowej na 400 milionach par obraz-tekst zmapowano pojęcia wizualne i tekstowe we wspólnej przestrzeni wektorowej. Umożliwiło to klasyfikację obrazów metodą zero-shot.

Współczesne konstrukcje (np. GPT-5.5, LLaMA 4) nie opierają się już na kaskadowym łączeniu osobnych enkoderów. Są od początku trenowane kompleksowo (ang. *end-to-end*) na danych wielomodalnych. Wpływa to korzystnie na interpretację wzajemnych powiązań między różnymi formami przekazu.

W analizie portali internetowych cechy te ułatwiają realizację zadań takich jak:

- interpretacja artykułów sieciowych łącznie z osadzonymi w nich infografikami, tabelami czy wykresami,
- przetwarzanie plików PDF o skomplikowanym układzie graficznym, trudnym do poprawnego odczytania tradycyjnymi parserami,
- bezpośrednia analiza materiałów wideo i ścieżek dźwiękowych bez czasochłonnej, wstępnej transkrypcji na postać tekstową.

2.4 Osadzenia zdań i dokumentów

Podczas gdy osadzenia słów pozwalają na analizę pojęciową na poziomie pojedynczych wyrazów, większość rzeczywistych systemów, w tym wyszukiwarki semantyczne i RAG, wymaga reprezentowania

znaczenia całych jednostek tekstowych – zdań, akapitów lub kompletnych dokumentów. W tej sekcji opisano proces przejścia od osadzeń słownych do zdaniowych, przedstawiono współczesne modele osadzeń oraz omówiono standardy ich ewaluacji.

2.4.1 Od osadzeń słów do osadzeń zdań

Najprostszy sposób generowania wektorów zdań stanowi uśrednianie statycznych osadzeń słów składowych (tzw. *mean pooling*). Podejście to pomija jednak szyk wyrazów i gramatykę. W konsekwencji wypowiedzi o różnym znaczeniu, lecz identycznym słownictwie (np. „pies pogryzł człowieka” oraz „człowiek pogryzł psa”), otrzymują tożsame reprezentacje.

Wprowadzenie modelu Sentence-BERT (SBERT) [27] usunęło te ograniczenia. Wcześniej porównywanie zdań przy użyciu klasycznego modelu BERT wymagało łączenia ich par tokenem separatora i realizowania pełnego przejścia w przód. Dla zbioru 10 000 zdań oznaczało to konieczność wykonania około 50 milionów operacji. Proces ten zajmował wówczas kilkadziesiąt godzin na standardowej infrastrukturze.

SBERT stosuje sieci syjamskie (ang. *siamese networks*) oraz trójkowe (ang. *triplet networks*). Każda wypowiedź trafia do enkodera osobno, po czym warstwa *mean pooling* buduje dla niej wektor o stałej długości. Uczenie sieci wykorzystuje funkcje kosztu oparte na klasyfikacji par, regresji podobieństwa lub stracie kontrastowej (ang. *contrastive loss*). Zapewnia to zbliżenie wektorów zdań o podobnym znaczeniu. Dzięki temu osadzenia można zapisywać niezależnie, a czas wyszukiwania w zbiorze 10 000 pozycji skrócił się do kilku sekund.

Powszechne zastosowanie w tym obszarze zyskały:

- **all-MiniLM-L6-v2**: lekki model o wymiarze wektora 384, gwarantujący optymalną szybkość przy dobrej precyzji semantycznej.
- **all-mpnet-base-v2**: wariant o wymiarze 768, oferujący wyższą dokładność kosztem dłuższego czasu obliczeń.
- **multilingual-e5-large**: model od firmy Microsoft, wspierający ponad 100 języków (w tym język polski), co pozwala na realizację wyszukiwania wielojęzycznego.

2.4.2 Współczesne modele osadzeń

Obecnie stosuje się zarówno rozwiązania chmurowe, jak i zaawansowane modele otwarte. Służą one do wyznaczania precyzyjnych reprezentacji pojęciowych. W grupie tej wyróżniają się:

- **OpenAI text-embedding-3-large**: komercyjny model generujący wektory o wymiarowości do 3072. Wprowadza technikę osadzeń warstwowych (ang. *matryoshka embeddings*), pozwalającą na bezpieczne skracanie wektorów (np. do 1536 lub 256) przy minimalnej utracie precyzji semantycznej.
- **Cohere Embed v3**: system dedykowany do wyszukiwania informacji (IR). Oferuje natywną kompresję wektorów do formatów int8 oraz binary, drastycznie obniżając zużycie pamięci RAM w bazach wektorowych.
- **BGE** (ang. *BAAI General Embedding*): otwarta rodzina modeli opracowana przez Pekinśki Instytut Sztucznej Inteligencji, notująca wysokie wyniki w globalnych rankingach wyszukiwania semantycznego.
- **E5** (ang. *Embeddings from bidirectional Encoder Representations*): modele szkolone kontrastowo na automatycznie generowanych parach tekstowych, cechujące się dobrymi zdolnościami uogólniania bez wcześniejszego dostrajania.

2.4.3 Benchmarki ewaluacyjne

Do obiektywnego porównywania modeli osadzeń służą zunifikowane środowiska testowe. Szeroko stosowanym standardem w wyszukiwaniu informacji pozostaje zestaw BEIR (*Benchmarking IR*) [28]. Weryfikuje on zachowanie modeli w trybie zero-shot na 18 zbiorach danych z różnych obszarów tematycznych, takich jak medycyna, nauka czy finanse. Pozwala to określić zdolności modelu do adaptacji dziedzinowej.

Z kolei MTEB (*Massive Text Embedding Benchmark*) stanowi najbardziej wszechstronny zestaw ewaluacyjny. Gromadzi on dziesiątki testów podzielonych na kilka głównych obszarów:

- klasyfikację tekstu oraz przypisywanie odpowiednich etykiet,
- klasteryzację, czyli grupowanie semantyczne wypowiedzi,
- klasyfikację par nakierowaną na ocenę relacji między zdaniami,
- ponowne ocenianie (ang. *reranking*),
- wyszukiwanie informacji (pobieranie dokumentów dla zapytania),
- podobieństwo semantyczne (ang. *Semantic Textual Similarity* – STS),
- streszczanie tekstu (ocena zgodności streszczeń z oryginałem).

Do najważniejszych metryk stosowanych w tych benchmarkach należą:

- **nDCG@k** (ang. *Normalized Discounted Cumulative Gain* na pozycji k): metryka oceniająca jakość rankingu wyników, biorąca pod uwagę pozycję relewantnych dokumentów – dokumenty bardziej istotne umieszczone wyżej w rankingu dają wyższą wartość metryki,
- **MAP** (ang. *Mean Average Precision*): średnia precyzja obliczana dla wielu zapytań, uwzględniająca kolejność zwracanych wyników,
- **MRR** (ang. *Mean Reciprocal Rank*): średnia z odwrotności pozycji pierwszego relewantnego dokumentu zwróconego przez system,
- **Recall@k**: odsetek wszystkich istotnych dokumentów, które system zdołał umieścić w pierwszych k wynikach wyszukiwania.

2.5 Bazy wektorowe i wyszukiwanie podobieństwa

Przechowywanie i przeszukiwanie wektorów wysokowymiarowych wymagało stworzenia wyspecjalizowanych baz danych. Klasyczne relacyjne lub dokumentowe systemy korzystają z indeksów typu B-drzewa (ang. *B-Trees*). Są one zoptymalizowane pod kątem wyszukiwania dokładnego. W obliczu problemów geometrycznych i wyszukiwania najbliższych sąsiadów wykazują się one jednak niską wydajnością. W niniejszym rozdziale scharakteryzowano architekturę baz wektorowych, popularne silniki bazodanowe, algorytmy indeksowania oraz stosowane metryki odległości.

2.5.1 Potrzeba baz wektorowych

Największą trudność w wyszukiwaniu wektorowym stanowi przekleństwo wymiarowości (ang. *curse of dimensionality*). W przestrzeniach wielowymiarowych relacje odległości ulegają spłaszczeniu. Tradycyjne indeksy jednowymiarowe tracą wtedy zdolności dyskryminacyjne. Z kolei precyzyjne wyszukiwanie k najbliższych sąsiadów (K-NN) metodą brute-force wymaga porównania zapytania z każdym

obiektom w bazie. Złożoność obliczeniowa takiego procesu wynosi $O(nd)$, gdzie n to liczba wektorów, a d odpowiada ich wymiarowości. Dla zbiorów o rozmiarach rzędu milionów pozycji uniemożliwia to pracę w czasie rzeczywistym.

Problem ten rozwiązują indeksy przybliżonego wyszukiwania najbliższych sąsiadów (ang. *Approximate Nearest Neighbor* – ANN). Skracają one czas wykonywania zapytań kosztem nieznacznego spadku dokładności. Dodatkowo nowoczesne bazy wektorowe muszą realizować:

- filtrowanie hybrydowe, łączące wyszukiwanie semantyczne z ograniczeniami na metadanych strukturalnych,
- dynamiczną aktualizację indeksów w czasie rzeczywistym bez wstrzymywania obsługi zapytań,
- skalowanie horyzontalne służące do dystrybucji danych w architekturze rozproszonej.

2.5.2 Popularne rozwiązania baz wektorowych

Dostępne na rynku rozwiązania bazodanowe różnią się architekturą, wydajnością oraz modelem wdrożenia. Do najpopularniejszych należą:

- **ChromaDB** [29]: otwarta baza zintegrowana z ekosystemem Pythona. Działa w trybie in-memory lub jako lekki system oparty na SQLite. Rozwiązanie to sprawdza się przy budowie prototypów ze względu na prostą konfigurację oraz natywną współpracę z pakietami LangChain czy LlamaIndex.
- **Pinecone** [30]: komercyjna baza danych udostępniana w modelu SaaS (ang. *Software as a Service*). Zwalnia użytkownika z konieczności zarządzania infrastrukturą serwerową. Zapewnia automatyczne skalowanie, niskie opóźnienia dla zbiorów miliardowych oraz sprawne filtrowanie metadanych.
- **Weaviate** [31]: otwartoźródłowy system zaprojektowany do realizacji wyszukiwania hybrydowego. Pozwala na bezpośrednie łączenie wektorów z klasycznym wyszukiwaniem słów kluczowych przy użyciu algorytmu BM25. Posiada gotowe konektory do integracji z modelami OpenAI, Cohere oraz HuggingFace.
- **Milvus** [32]: rozproszone rozwiązanie klasy enterprise, przystosowane do pracy z miliardowymi zbiorami danych. Charakteryzuje się modułową budową dzielącą operacje zapisu, odczytu i generowania indeksów na osobne komponenty. Wspiera akcelerację sprzętową GPU.
- **Qdrant** [33]: silnik wyszukiwania napisany w języku Rust. Wybór tej technologii gwarantuje wysoką stabilność, szybkość operacji i oszczędne zarządzanie pamięcią operacyjną. Qdrant umożliwia filtrowanie metadanych (ang. *payload filtering*) oraz natywną obsługę wektorów rzadkich i gęstych.
- **FAISS** [34]: biblioteka stworzona przez Meta AI. Nie stanowi ona kompletnej bazy danych, lecz zbiór zoptymalizowanych algorytmów napisanych w C++ (z interfejsem dla języka Python). Rozwiązanie to służy jako fundament silników indeksujących w wielu komercyjnych i otwartych bazach danych.

2.5.3 Algorytmy indeksowania

Podstawą organizacji przestrzeni wektorowej są algorytmy ANN. Służą one do szybkiego wyszukiwania przybliżonego. Do najważniejszych z nich należą:

- **HNSW** (ang. *Hierarchical Navigable Small World*) [35]: graf hierarchiczny łączący wektory w wielowarstwową strukturę. Najwyższe warstwy pełnią rolę list pomijanych (ang. *skip lists*), pozwalającą na szybką nawigację do ogólnego obszaru semantycznego. Warstwy dolne zawierają gęsty graf służący do precyzyjnego dostrojenia wyniku. HNSW cechuje wysoka celność (Recall powyżej 99%) i krótki czas wyszukiwania, wymaga jednak znacznych ilości pamięci operacyjnej do zapisu struktury grafu.
- **IVF** (ang. *Inverted File Index*): technika dzieląca przestrzeń wektorową na klastry przy użyciu algorytmu *k*-średnich (ang. *k-means*). Każdy wektor zostaje przypisany do najbliższego centroidu. Wyszukiwanie ogranicza się do analizy elementów w obrębie najbliższych klastrów. Rozwiązanie to przyspiesza budowę indeksu i oszczędza pamięć RAM, lecz oferuje nieco niższą celność niż HNSW.
- **LSH** (ang. *Locality-Sensitive Hashing*): podejście probabilistyczne oparte na funkcjach haszujących, które odwzorowują wektory bliskie semantycznie w te same kubelki danych. LSH charakteryzuje się dużą szybkością działania, ale najniższą dokładnością, co ogranicza jego zastosowanie do deduplikacji danych.
- **Kwantyzacja produktowa** (ang. *Product Quantization* – PQ): kompresja wektorów polegająca na podziale przestrzeni na niezależne podprzestrzenie i ich klastrowaniu. PQ redukuje wymagania pamięciowe nawet o 90% – 99%. W celach skalowalności łączy się ją z IVF (wariant IVF-PQ).

Zestawienie i porównanie omówionych algorytmów indeksowania przedstawiono w Tabeli 2.3.

Tabela 2.3: Porównanie głównych algorytmów indeksowania wektorowego

Algorytm	Szybkość wyszukiwania	Zużycie pamięci RAM	Dokładność (Recall)
FLAT (K-NN)	Bardzo niska ($O(n)$)	Bardzo niskie	Idealna (100%)
HNSW	Bardzo wysoka	Wysokie (struktura grafu)	Bardzo wysoka (> 99%)
IVF	Wysoka	Średnie	Średnio-wysoka
IVF-PQ	Bardzo wysoka	Bardzo niskie (kompresja)	Średnia
LSH	Bardzo wysoka	Niskie	Niska do średniej

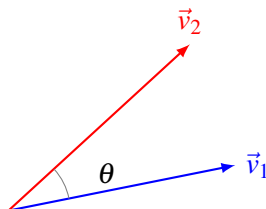
2.5.4 Metryki odległości

Wybór odpowiedniej metryki odległości decyduje o poprawności wyszukiwania semantycznego, ponieważ musi być ona zgodna z miarą przyjętą na etapie uczenia modelu osadzeń:

- **Podobieństwo cosinusowe** (ang. *Cosine Similarity*): mierzy cosinus kąta między dwoma wektorami, ignorując ich długość (magnitudę). Przyjmuje wartości z przedziału $[-1, 1]$, gdzie 1 oznacza identyczny kierunek wektorów. Jest to najczęściej wybierana metryka w zadaniach NLP, ponieważ doskonale radzi sobie z tekstami o różnej długości.
- **Odległość euklidesowa (L2)**: klasyczna odległość geometryczna w linii prostej między dwoma punktami w przestrzeni wielowymiarowej. Jest wrażliwa na różnice w długości wektorów, co może negatywnie wpływać na wyniki w przypadku porównywania tekstów o skrajnie różnych rozmiarach.

- **Iloczyn skalarny** (ang. *Dot Product*): prosta suma iloczynów odpowiadających sobie współrzędnych obu wektorów. Obliczeniowo jest to najszybsza metryka, która w przypadku wektorów znormalizowanych jednostkowo daje wyniki tożsame z podobieństwem cosinusowym.

Rysunek 2.2 przedstawia geometryczną interpretację podobieństwa cosinusowego jako kąta θ pomiędzy dwoma wektorami osadzeń w przestrzeni semantycznej.



Rysunek 2.2: Podobieństwo cosinusowe jako kąt θ między dwoma wektorami osadzeń; im mniejszy kąt, tym większe podobieństwo semantyczne.

2.6 Generowanie wspomagane wyszukiwaniem (RAG)

Generowanie wspomagane wyszukiwaniem (ang. *Retrieval-Augmented Generation* – RAG) stanowi obecnie wiodący paradygmat projektowania systemów dialogowych i analitycznych opartych na dużych modelach językowych. Łączy ono zalety zewnętrznych, nieparametrycznych baz wiedzy ze zdolnościami modeli generatywnych. Alternatywę dla tego podejścia stanowi umieszczanie całych dokumentów bezpośrednio w oknie kontekstowym modeli o bardzo dużej pojemności (np. Gemini 2.5 Pro czy Claude Sonnet 4.6, opisanych w podrozdziale 2.3). Badania porównawcze wskazują jednak, że samo powiększanie kontekstu nie zastępuje ukierunkowanego wyszukiwania: modele te tracą na precyzji wraz ze wzrostem ilości podanych, często nieistotnych informacji, a koszt i czas przetwarzania rosną proporcjonalnie do rozmiaru kontekstu [36]. RAG pozostaje więc bardziej efektywnym rozwiązaniem przy analizie dużych, zmieniających się zbiorów treści internetowych. W rozdziale tym przedstawiono motywację do stosowania tej metody, architekturę potoku przetwarzania, strategie podziału tekstu, zaawansowane techniki optymalizacji oraz kryteria ewaluacji systemów klasy RAG.

2.6.1 Motywacja i koncepcja

Stosowanie podejścia RAG wynika z dążenia do eliminacji podstawowych ograniczeń modeli LLM. Wśród nich wymienia się przede wszystkim:

- **Zjawisko halucynacji:** generowanie nieprawdziwych, lecz brzmiących wiarygodnie faktów.
- **Zamrożona wiedza:** brak dostępu do informacji opublikowanych po dacie zakończenia treningu sieci (tzw. *knowledge cutoff*).
- **Brak dostępu do zasobów prywatnych:** modele nie posiadają wiedzy o wewnętrznych bazach danych ani dynamicznych serwisach sieciowych.
- **Wysoki koszt dostrajania:** aktualizacja wiedzy modeli za pomocą fine-tuningu jest kosztowna obliczeniowo i grozi katastrofalnym zapominaniem (ang. *catastrophic forgetting*).

Integracja dynamicznego wyszukiwania z generowaniem tekstu [8] poprawiła wierność faktograficzną modeli. Wyszukane dokumenty są podawane bezpośrednio w oknie kontekstowym. Zadaniem sieci LLM staje się wówczas wyłącznie analiza, synteza oraz ustrukturyzowanie odpowiedzi na podstawie dostarczonych materiałów. Nawet przy modelach dysponujących oknem kontekstowym rzędu miliona tokenów

okno to pozostaje ograniczone względem rozmiaru rzeczywistych zasobów internetowych. Nie jest więc możliwe umieszczenie w promptcie całej, stale rosnącej treści serwisu. Dodatkowym ograniczeniem pozostaje zjawisko „zagubienia w środku” (ang. *Lost in the Middle*) [37]: modele LLM skuteczniej korzystają z informacji umieszczonych na początku i końcu kontekstu niż z tych położonych centralnie. Z tego powodu selektywne wyszukiwanie fragmentów pozostaje istotnym uzupełnieniem samego powiększania okna kontekstowego.

2.6.2 Architektura potoku RAG

Uproszczony schemat dwufazowej architektury systemu RAG przedstawiono na Rysunku 2.3. Systemy RAG opierają się na dwufazowej strukturze przetwarzania, obejmującej fazę indeksowania danych (realizowaną offline) oraz fazę wyszukiwania i generowania odpowiedzi (realizowaną online).

Poszczególne etapy przebiegają następująco:

1. Faza indeksowania (offline):

- Pozyskiwanie danych poprzez pobieranie dokumentów z wybranych źródeł (np. scraping serwisów internetowych).
- Ekstrakcja tekstu polegająca na usunięciu elementów strukturalnych z formatów HTML czy PDF.
- Segmentacja (ang. *chunking*), czyli podział długich tekstów na spójne tematycznie fragmenty.
- Generowanie wektorów semantycznych dla każdego fragmentu przy użyciu modelu osadzeń.
- Zapis wektorów wraz z metadanymi (np. adresem źródłowym URL czy tytułem) w bazie wektorowej.

2. Faza wyszukiwania i generowania (online):

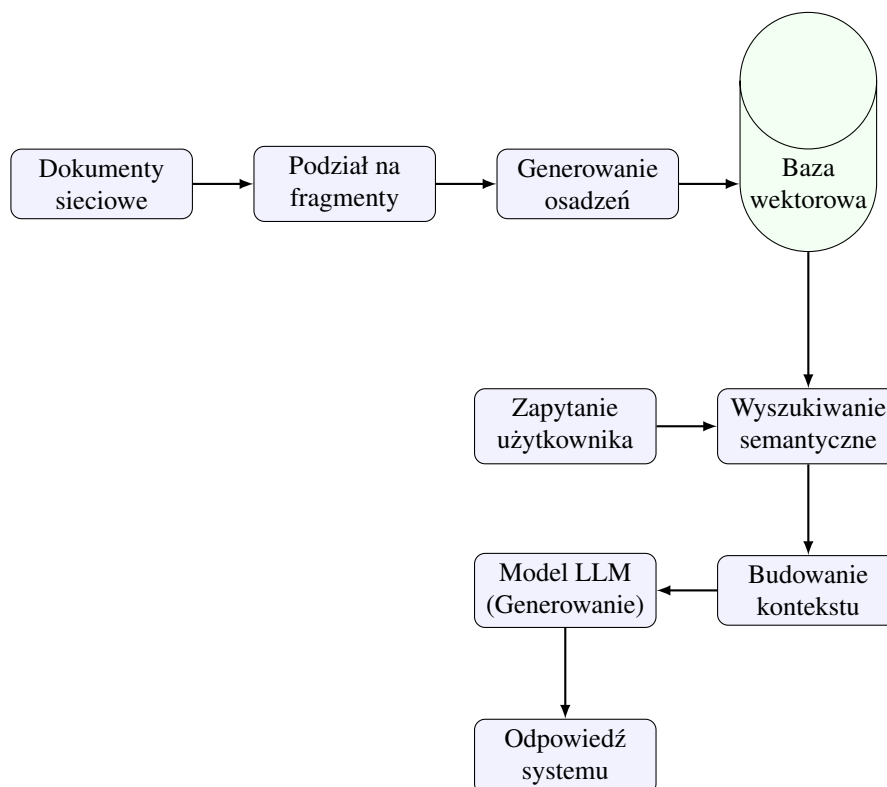
- Konwersja pytania użytkownika na wektor semantyczny za pomocą tego samego modelu osadzeń.
- Pobranie z bazy danych k najbardziej podobnych fragmentów tekstu na podstawie wybranej miary odległości.
- Zbudowanie ustrukturyzowanego promptu łączącego pobrany kontekst z zapytaniem.
- Przekazanie promptu do modelu LLM w celu wygenerowania odpowiedzi ugruntowanej w faktach.

2.6.3 Strategie dzielenia tekstu na fragmenty

Sposób podziału dokumentów (ang. *chunking*) rzutuje bezpośrednio na jakość wyszukiwania semantycznego. Zbyt małe bloki tekstu nie niosą pełnego kontekstu. Z kolei bloki nadmiernie rozbudowane rozmywają sygnał pojęciowy i mogą wykraczać poza okno kontekstowe modelu językowego.

W praktyce stosuje się następujące metody podziału:

- **Podział o stałym rozmiarze** (ang. *Fixed-Size Chunking*): polega na krojeniu tekstu na bloki o określonej liczbie tokenów bądź znaków z użyciem zakładki (ang. *overlap*) rzędu 10% – 20%. Zapobiega to utracie kontekstu na granicach podziału. Metoda ta jest prosta, lecz może rozrywać zdania w losowych miejscach.



Rysunek 2.3: Uproszczony schemat dwufazowej architektury systemu RAG: indeksowanie offline (górny potok) oraz wyszukiwanie i generowanie online (dolny potok)

- **Podział semantyczny** (ang. *Semantic Chunking*): uwzględnia strukturę wypowiedzi. Cięcia realizuje się na granicach zdań lub akapitów przy wykorzystaniu separatorów strukturalnych. Zaawansowane podejścia badają różnicę podobieństwa osadzeń kolejnych zdań i wyznaczają punkty podziału tam, gdzie podobieństwo to gwałtownie spada, co sygnalizuje zmianę tematu.
- **Podział ustrukturyzowany** (ang. *Document-Aware Chunking*): respektuje strukturę dokumentów (np. HTML lub Markdown). Nagłówki wyznaczają granice segmentów, a tabele czy listy są traktowane jako niepodzielne, atomowe bloki tekstu, co chroni ich spójność znaczeniową.

2.6.4 Zaawansowane techniki RAG

W celu wyeliminowania wad podstawowej, naiwnej architektury (ang. *Naive RAG*) opracowano zaawansowane metody optymalizacji [38]. Do najistotniejszych usprawnień należą:

- **Modyfikacja zapytań** (ang. *Query Enhancement*):
 - Przepisywanie zapytań (ang. *Query Rewriting*), gdzie LLM tłumaczy skrótowe lub nieprecyzyjne pytania na formę bardziej zdatną do wyszukiwania.
 - Ekspansja zapytań (ang. *Query Expansion*) polegająca na generowaniu wariantów synonimicznych pytania bazowego.
 - Metoda HyDE (ang. *Hypothetical Document Embeddings*), w której model LLM tworzy sztuczną, uproszczoną odpowiedź na zapytanie użytkownika. Dopiero jej wektor osadzenia służy do odpytania bazy, co ułatwia wyszukiwanie, gdyż wektory odpowiedzi leżą bliżej dokumentów źródłowych niż wektory pytań.

- **Optymalizacja pobierania danych** (ang. *Retrieval Optimization*):
 - Wyszukiwanie hybrydowe, łączące wyszukiwanie gęste (semantyczne) z rzadkim (BM25). Pozwala to łączyć zalety rozumienia pojęciowego z dokładnym wyszukiwaniem kodów, dat czy nazw własnych.
 - Mechanizm rodzic-dziecko (ang. *Parent-Child Chunking*), w którym baza przechowuje małe, precyzyjne fragmenty (dzieci) powiązane z większymi blokami tekstu (rodzicami). Wyszukiwanie wektorowe identyfikuje mały fragment, lecz do modelu LLM przekazywany jest pełny kontekst rodzica, co zapewnia precyzję wyszukiwania bez utraty kontekstu generowania.
- **Podejścia hierarchiczne**: model RAPTOR [39] polegający na rekurencyjnym grupowaniu i streszczaniu fragmentów dokumentów przy użyciu LLM, co tworzy strukturę drzewiastą o różnych poziomach abstrakcji. Umożliwia to efektywne odpowiadanie na pytania ogólne, wymagające syntezy wiedzy z całego dokumentu, a nie tylko z jego małego wycinka.
- **Ponowne ocenianie** (ang. *Reranking*): dwuetapowa procedura wyszukiwania. Najpierw baza wektorowa szybko pobiera np. 100 dokumentów przy użyciu prostego podobieństwa wektorów. Następnie kosztowny obliczeniowo, lecz precyzyjny model typu Cross-Encoder dokonuje ostatecznej selekcji i ustala ranking dla k najlepszych dokumentów.
- **Optymalizacja okna kontekstowego**: zapobiega zjawisku „zagubienia w środku” (ang. *Lost in the Middle*), czyli tendencji modeli LLM do ignorowania informacji umieszczonych w środkowej części promptu. Rozwiązaniem jest sortowanie dokumentów tak, by kluczowe fragmenty trafiały na skrajne pozycje, lub kompresja kontekstu w celu eliminacji szumu.

2.6.5 Ewaluacja systemów RAG

Ewaluacja systemów RAG jest złożona, ponieważ wymaga niezależnej oceny modułu wyszukiwania (ang. *retrieval*) oraz modułu generowania (ang. *generation*). Standardowe metryki takie jak BLEU [40] czy ROUGE [41] okazały się niewystarczające, ponieważ oceniają jedynie podobieństwo tekstowe (n-gramowe) do odpowiedzi referencyjnej, pomijając poprawność merytoryczną i semantyczną.

Współczesny standard ewaluacji opiera się na tzw. triadzie RAG, implementowanej przez automatyczne frameworki takie jak RAGAS [42] czy TruLens [43]. Triada ta definiuje trzy podstawowe kryteria oceny realizowanej z wykorzystaniem silniejszego modelu LLM (np. GPT-5.5 lub Claude Sonnet) w roli sędziego (ang. *LLM-as-a-judge*):

- **Precyzja kontekstu** (ang. *Context Precision*): ocenia, czy pobrane fragmenty dokumentów rzeczywiście zawierają informacje niezbędne do udzielenia odpowiedzi na zapytanie użytkownika,
- **Wierność/Wiarygodność** (ang. *Faithfulness*): mierzy stopień ugruntowania wygenerowanej odpowiedzi w dostarczonym kontekście – pozwala to na precyzyjne wykrywanie i eliminowanie halucynacji (odpowiedź zawierająca fakty nieobecne w kontekście otrzymuje niską ocenę),
- **Relevantność odpowiedzi** (ang. *Answer Relevance*): ocenia, czy wygenerowany tekst bezpośrednio odpowiada na pytanie postawione przez użytkownika, bez uciekania od tematu lub lania wody.

W Listingu 2.1 przedstawiono klasyczny szablon promptu RAG, który instruuje model LLM do rygorystycznego trzymania się dostarczonego kontekstu, minimalizując ryzyko wystąpienia halucynacji.

Listing 2.1: Przykładowy szablon promptu integrujący pobrany kontekst z zapytaniem użytkownika

```

1 # Szablon promptu RAG w języku Python
2 PROMPT_TEMPLATE = """

```

```
3 Jesteś pomocnym asystentem analitycznym. Twoim zadaniem jest udzielenie
   odpowiedzi na pytanie użytkownika wyłącznie na podstawie
   dostarczonego kontekstu.
4 Jeśli w kontekście nie ma informacji pozwalających na odpowiedź, napisz
   szczerze, że nie posiadasz takich informacji.
5
6 Kontekst:
7 {context}
8
9 Pytanie:
10 {question}
11
12 Odpowiedź:
13 ""
```

2.7 Zastosowania w analizie treści internetowych

Metody reprezentacji semantycznej, systemy LLM oraz architektury RAG wspierają automatyzację analizy zasobów sieciowych. Witryny internetowe stanowią wyzwanie badawcze. Są one nieustrukturyzowane, dynamiczne i pełne szumu informacyjnego. W tym rozdziale opisano trudności związane z ekstrakcją danych, podstawowe zadania przetwarzania semantycznego oraz korzyści płynące ze stosowania podejścia semantycznego.

2.7.1 Ekstrakcja treści ze stron internetowych

Pozyskiwanie danych ze stron WWW (ang. *web scraping*) otwiera potok analityczny [44]. Prace te utrudniają bariery techniczne, do których należą:

- Zróżnicowana i niestabilna struktura dokumentów HTML w poszczególnych serwisach.
- Renderowanie zawartości po stronie klienta (frameworki JavaScript, np. React czy Angular), co wymusza użycie przeglądarek bezgłowych (ang. *headless browsers*).
- Systemy ochrony przed botami (np. Cloudflare, testy CAPTCHA) blokujące masowe pobieranie danych.
- Częste modyfikacje układu graficznego (layoutu) niszczące działanie klasycznych parserów opartych na selektorach CSS lub ścieżkach XPath.

Istotnym krokiem przetwarzania HTML jest czyszczenie tekstu z elementów pobocznych (ang. *boilerplate removal*). Strony WWW zawierają m.in. menu, reklamy, komentarze oraz stopki. Elementy te stanowią szum, który pogarsza jakość osadzeń semantycznych. Do ich odrzucania stosuje się heurystyki (np. bibliotekę Readability badającą gęstość tekstu w drzewie DOM) lub klasyfikatory uczenia maszynowego selekcionujące właściwy artykuł.

Ważne zadanie stanowi również ekstrakcja ustrukturyzowana. Serwisy internetowe często wdraża mikrodane Schema.org (np. format JSON-LD), dostarczające gotowych metadanych o autorze, dacie publikacji czy ocenach. Równoległe algorytmy ekstrakcji muszą dbać o zachowanie struktury tabel oraz list, gdyż spłaszczenie ich do surowego tekstu niszczy relacje między danymi.

2.7.2 Zadania analizy semantycznej

Wyodrębnienie czystego tekstu pozwala na realizację zaawansowanych zadań analitycznych. Do podstawowych kierunków należą:

- **Klasyfikacja tematyczna:** kategoryzacja artykułów (np. sport, biznes). Stosuje się podejścia wieloetykietowe (ang. *multi-label classification*) oraz taksonomie hierarchiczne gwarantujące precyzyjne mapowanie wątków.
- **Analiza sentymentu** (ang. *Sentiment Analysis*): badanie zabarwienia emocjonalnego wypowiedzi. Klasyczne podejścia dokumentowe zastępuje się analizą aspektową (ang. *Aspect-Based Sentiment Analysis* – ABSA) określającą emocje wobec cech obiektów (np. ocena ekranu i baterii w telefonie). Odrębnym nurtem jest detekcja postaw (ang. *stance detection*) wobec kontrowersyjnych zjawisk.
- **Rozpoznawanie i mapowanie encji** (NER & Linking): detekcja nazw własnych (osób, firm, państw) i ich powiązanie (ang. *Entity Linking*) ze słownikami pojęciowymi (Wikidata, DBpedia). Pozwala to na rozróżnianie homonimów (np. marka Apple a owoc jabłko).
- **Streszczanie tekstu** (ang. *Summarization*): skracanie artykułów. Metody ekstrakcyjne wybierają kluczowe zdania z oryginału. Z kolei techniki abstrakcyjne (ang. *abstractive summarization*) wykorzystują potencjał generatywny LLM do formułowania nowych, zwiezłych syntez.

2.7.3 Odpowiadanie na pytania dotyczące treści internetowych

Interakcja w języku naturalnym z bazami wiedzy stanowi jedno z najważniejszych zastosowań metod NLP. Historycznie podejście to ewoluowało od systemów ekstrakcyjnych (wycinanie fragmentu tekstu źródłowego) do abstrakcyjnych, generujących swobodne wypowiedzi.

Wdrożenie architektury RAG w analizie portali sieciowych napotyka specyficzne wyzwania:

- **Rozbieżność informacji:** różne serwisy mogą odmiennie przedstawiać te same fakty. System musi identyfikować takie konflikty i prezentować je odbiorcy.
- **Weryfikacja wiarygodności i atrybucja:** treści internetowe bywają nieprawdziwe lub nieaktualne. Wymaga to ważenia autorytetu źródeł oraz rygorystycznego cytowania linków źródłowych (ang. *attribution*) w celu umożliwienia samodzielnej weryfikacji przez czytelnika.
- **Zarządzanie niepewnością:** w przypadku braku odpowiedzi w zebranych dokumentach system musi powstrzymać się od generowania treści, co zapobiega halucynacjom.

2.7.4 Przewagi nad wyszukiwaniem słów kluczowych

Wyszukiwanie semantyczne niesie istotne korzyści względem indeksów odwróconych. Do głównych wyróżników należą:

- **Zrozumienie znaczenia:** systemy semantyczne rozwiązują problem synonimii (zapytanie o „rekrutację pracowników” poprawnie wiąże się ze sformułowaniem „zatrudnianie personelu”) oraz homonimii (interpretacja znaczenia słów wieloznacznych na podstawie otoczenia językowego).
- **Analiza intencji i negacji:** tradycyjne wyszukiwarki słów kluczowych ignorują zaprzeczenia oraz relacje syntaktyczne. Fraza „leki bez interakcji z alkoholem” w klasycznym ujęciu zwróci opisy interakcji, podczas gdy wyszukiwanie semantyczne poprawnie odczyta rolę przyminka „bez”.
- **Synteza odpowiedzi:** system analityczny automatycznie agreguje informacje z wielu stron, odrzuca powtórzenia i tworzy gotową, zwiezłą odpowiedź zamiast listy setek odnośników do samodzielnej analizy.
- **Interakcja wielotururowa:** architektura umożliwia prowadzenie ciągłego dialogu (ang. *multi-turn conversation*). Użytkownik może zadawać pytania doprecyzowujące, a system poprawnie odnosi zaimki do wątków poruszonych we wcześniejszych turach rozmowy.

2.8 Podsumowanie rozdziału

W niniejszym rozdziale dokonano szczegółowego przeglądu stanu wiedzy z zakresu przetwarzania języka naturalnego. Przejście od rzadkich reprezentacji symbolicznych (Bag-of-Words, TF-IDF), przez statyczne osadzenia słowne, aż do głębokich enkoderów opartych na architekturze Transformer, zrewolucjonizowało rozumienie semantyki tekstu. Maszyny zyskały zdolność analizy subtelnych kontekstów językowych.

Rozpatrzone systemy LLM, zarówno komercyjne (np. GPT-5.5 czy Claude), jak i otwarte (LLaMA, Mistral), wykazują wysokie zdolności wnioskowania. Są one jednak podatne na halucynacje. Ich wiedza parametryczna pozostaje zamrożona. Słabości te eliminuje wdrożenie architektury RAG. Mechanizmy te opierają się na bazach wektorowych (Qdrant, Pinecone, Weaviate) i indeksowaniu przybliżonym (głównie algorytmy HNSW oraz kwantyzacja produktowa). Umożliwiają one zasilenie promptu wejściowego sprawdzonymi, zewnętrznymi faktami pobieranymi w czasie rzeczywistym.

W obszarze analizy treści sieciowych połączenie to zmienia paradygmat pracy z informacją. Wyszukiwanie słów kluczowych ustępuje miejsca automatycznemu oczyszczaniu kodu HTML, segmentacji pojęciowej i bezpośredniej syntezie odpowiedzi. Opracowane wnioski stanowią podbudowę teoretyczną i metodologiczną dla dalszych prac. Następny rozdział szczegółowo opisuje architekturę, implementację oraz procedury ewaluacyjne projektowanego systemu analitycznego.

Rozdział 3

Założenia i metodologia proponowanego rozwiązania

Rozdział ten opisuje założenia badawcze, architekturę systemu, ramy metodologiczne oraz podejście ewaluacyjne. Opracowany system analizuje treści internetowe pod kątem semantycznego wyszukiwania i precyzyjnego odpowiadania na pytania użytkowników przy użyciu techniki generowania wspomaganego wyszukiwaniem (ang. *Retrieval-Augmented Generation*, RAG). Opisano tu ramy badawcze Design Science Research (DSR) oraz potoki danych. Przedstawiono metodologię przygotowania danych oraz empirycznej ewaluacji opartej na ośmiu konfiguracjach eksperymentalnych (C1–C8). Omówiono także kwestie etyczne i ograniczenia techniczne.

3.1 Podejście badawcze

Badania w obszarze inżynierii oprogramowania wymagają precyzyjnego sformułowania pytań badawczych i wyboru ram metodologicznych. Służy to usystematyzowaniu eksperymentów oraz ocenie przydatności powstałego artefaktu.

3.1.1 Pytania badawcze

Do oceny skuteczności i wydajności proponowanego systemu posłużą cztery pytania badawcze (ang. *Research Questions*, RQ):

RQ1: Jak różne podejścia do osadzania tekstu (ang. *sentence embeddings*) wpływają na jakość semantycznej reprezentacji treści stron internetowych?

Analizanie poddano wpływ zaawansowanych modeli osadzeń (np. Sentence-BERT [27]) na dokładność dopasowania semantycznego. Istotne jest porównanie ich z prostszymi, mniej wymagającymi obliczeniowo metodami.

RQ2: Jaka konfiguracja parametrów potoku RAG jest optymalna dla zadania odpowiadania na pytania na podstawie artykułów internetowych?

Badana jest zależność między rozmiarem fragmentu tekstu (ang. *chunk size*), nakładaniem się segmentów (ang. *overlap*) oraz liczbą wyszukiwanych dokumentów (k). Parametry te bezpośrednio decydują o poprawności merytorycznej odpowiedzi generowanych przez model LLM.

RQ3: Jaki jest wpływ poszczególnych komponentów architektury na ostateczną skuteczność systemu?

Badania ablacyjne (ang. *ablation studies*) pozwalają na wyizolowanie wpływu poszczególnych elementów architektury. Ocenie podlega wybór generatora, modelu osadzeń oraz rozmiar okna kontekstowego.

RQ4: Jakie są praktyczne wyzwania i uwarunkowania wdrożeniowe proponowanego rozwiązania w warunkach rzeczywistych?

Konieczna jest analiza czasu odpowiedzi systemu (ang. *latency*), zużycia pamięci, obciążenia CPU/GPU oraz kosztów zewnętrznych API. Czynniki te decydują o skalowalności rozwiązania.

3.1.2 Ramy metodologiczne

Proces badawczy osadzono w metodologii *Design Science Research* (DSR) [45]. Paradygmat ten polega na projektowaniu i ewaluacji artefaktów rozwiązujących konkretne problemy naukowe i inżynierskie. Pracę zorganizowano wokół sześciu etapów DSR:

1. **Identyfikacja problemu i motywacja:** Analiza barier w wyszukiwaniu semantycznym i śledzeniu dynamicznych zmian na stronach internetowych. Tradycyjne wyszukiwanie słów kluczowych zawodzi przy dużych wolumenach danych. Motywuje to do wdrożenia modeli generatywnych.
2. **Zdefiniowanie celów rozwiązania:** Określenie wymagań funkcjonalnych. Do celów należą: precyzyjne odnajdywanie kontekstu, redukcja halucynacji LLM oraz czas odpowiedzi poniżej 5 sekund.
3. **Projektowanie i rozwój:** Zbudowanie modułowej architektury i implementacja prototypu w języku Python. Rozwiązanie integruje asynchroniczny potok indeksowania (offline) oraz potok obsługi zapytań (online).
4. **Demonstracja:** Uruchomienie systemu na rzeczywistych artykułach prasowych i technicznych. Pozwala to dowiedzieć, że procesy pobierania, indeksowania i generowania odpowiedzi przebiegają prawidłowo.
5. **Ewaluacja:** Rygorystyczne testy empiryczne ośmiu konfiguracji (C1–C8). Analizie podlegają metryki wyszukiwania, czasy opóźnień oraz poprawność merytoryczna syntezy.
6. **Komunikacja:** Przedstawienie struktury i wniosków w niniejszej rozprawie oraz upublicznienie kodu źródłowego na potrzeby dalszych badań.

Podejście badawcze łączy metody ilościowe i jakościowe. Pomiary opóźnień i metryki wyszukiwania zestawiono z analizą błędów syntezy tekstu. Umożliwia to wszechstronną ocenę systemu. Prace prowadzono iteracyjnie, wykorzystując wyniki kolejnych testów do optymalizacji parametrów potoku RAG.

3.2 Projekt architektury systemu

Architektura modułowa ułatwia konserwację i testowanie poszczególnych elementów systemu. Umożliwia to również bezproblemową modyfikację kodu. Opisano tu architekturę, potoki danych oraz schematy przepływu informacji.

3.2.1 Ogólny projekt systemu

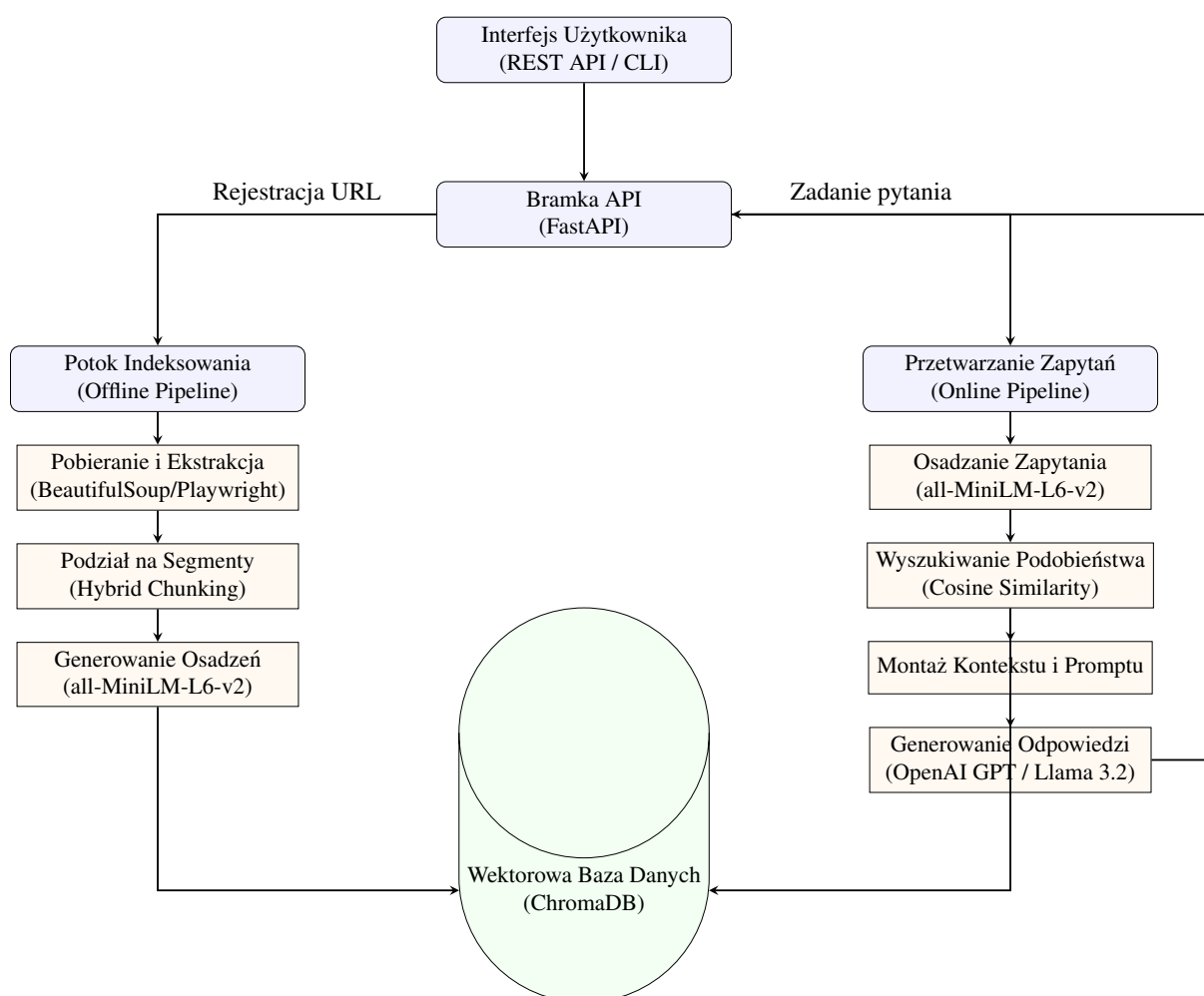
Pięć luźno powiązanych warstw funkcjonalnych tworzy szkielet systemu. Taki podział pozwala na niezależną wymianę elementów (np. bazy wektorowej) bez ingerencji w pozostałe moduły. Strukturę tworzą:

- **Warstwa Prezentacji (Interfejs Użytkownika):** REST API oraz konsola CLI (ang. *Command Line Interface*). Służą do przekazywania adresów URL i wprowadzania pytań.
- **Bramka API (FastAPI):** Pośrednik komunikacyjny. Waliduje dane wejściowe za pomocą biblioteki Pydantic, obsługuje wywołania asynchroniczne i rozdziela zadania.
- **Potok Indeksowania Treści (Offline Pipeline):** Odpowiada za pobieranie i oczyszczanie stron z szumów takich jak reklamy czy stopki. Segmentuje tekst (chunking) i tworzy wektory semantyczne.

- **Potok Przetwarzania Zapytań (Online Pipeline):** Przekształca zapytania w wektory, przeszukuje bazę, kompletuje kontekst oraz nadzoruje syntezę odpowiedzi przez model LLM.
- **Warstwa Składowania Danych (Wektorowa Baza Danych):** ChromaDB przechowująca osadzenia semantyczne wraz z metadanymi. Pozwala na szybkie wyszukiwanie najbliższych sąsiadów (ang. *Approximate Nearest Neighbor*, ANN).

Projekt spełnia cztery zasady inżynierii oprogramowania. Pierwszą jest modularność oparta na ściśle zdefiniowanych interfejsach. Drugą to separacja odpowiedzialności — asynchroniczne indeksowanie nie wpływa na obsługę zapytań online. Trzecią zasadę stanowi bezstanowość, co pozwala na łatwe skalowanie poziome aplikacji. Ostatnią jest trwałość danych realizowana przez ChromaDB, co zapobiega powtórному pobieraniu i wektoryzacji witryn.

Rysunek 3.1 przedstawia schemat blokowy ogólnej architektury modułowej zaproponowanego systemu, ilustrujący relacje pomiędzy poszczególnymi warstwami.



Rysunek 3.1: Schemat blokowy ogólnej architektury modułowej proponowanego systemu.

3.2.2 Potok indeksowania treści

Asynchroniczny potok indeksowania przetwarza surowy kod HTML w gęste reprezentacje wektorowe. Proces ten składa się z siedmiu etapów:

1. **Walidacja i Normalizacja URL:** Sprawdzenie poprawności zapisu adresu (protokół, domena). Usunięcie parametrów śledzących zapobiega duplikacji wpisów w bazie.
2. **Pobieranie Zawartości HTML:** Wykonanie żądania HTTP GET przy użyciu nagłówka User-Agent. Potok obsługuje błędy sieciowe, limity czasu (ang. *timeout*) oraz dławienie zapytań (ang. *rate limiting*).
3. **Parsowanie HTML i Ekstrakcja Treści:** Przekształcenie kodu HTML w strukturę drzewiastą. System odrzuca nawigację, panele boczne, reklamy i stopki, skupiając się na tagach `<article>`. Jednocześnie pobierane są metadane: tytuł, autor czy data publikacji.
4. **Przetwarzanie Wstępne Tekstu:** Normalizacja Unicode (NFC), usunięcie nadmiarowych białych znaków oraz usunięcie znaków sterujących. Zapobiega to zaburzeniom tokenizacji.
5. **Segmentacja Tekstu (Chunking):** Podział tekstu na mniejsze bloki. Zastosowanie podziału opartego na granicach zdań z kontrolowanym marginesem nakładania zapobiega utracie spójności semantycznej.
6. **Generowanie Osadzeń (Wektoryzacja):** Przekazanie fragmentów tekstu do modelu osadzeń (np. all-MiniLM-L6-v2), generującego gęste wektory. Przetwarzanie wsadowe (ang. *batch processing*) skraca czas obliczeń.
7. **Zapis w Wektorowej Bazie Danych:** Zapis wektorów, oryginalnego tekstu oraz metadanych w bazie ChromaDB. Następuje wtedy aktualizacja indeksu HNSW [35].

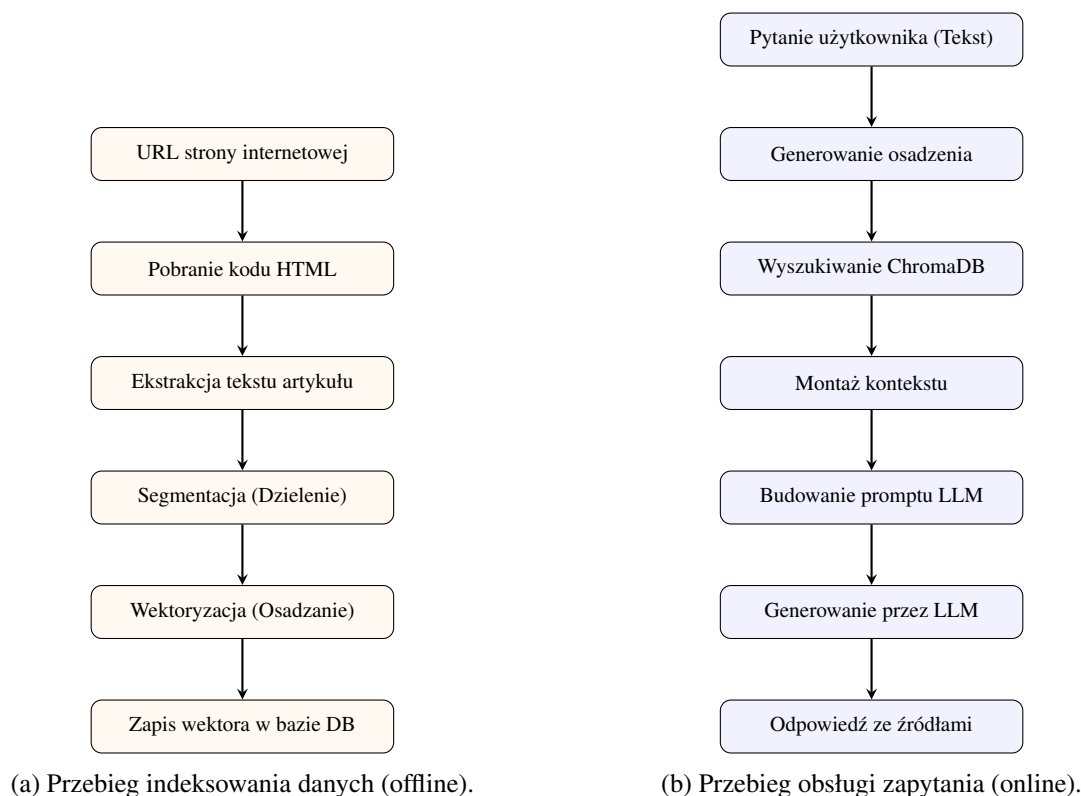
3.2.3 Potok przetwarzania zapytań

Interakcja z użytkownikiem odbywa się w czasie rzeczywistym. Zadaniem potoku online jest synteza odpowiedzi przy użyciu zgromadzonych zasobów. Proces ten przebiega według następującego schematu:

1. **Odbiór Zapytania i Walidacja:** Pobranie pytania w języku naturalnym i ewentualnych filtrów (np. ograniczenie wyszukiwania do jednego adresu URL).
2. **Osadzanie Zapytania:** Konwersja pytania na wektor semantyczny za pomocą modelu użytego przy indeksowaniu. Gwarantuje to spójność przestrzeni wektorowej.
3. **Wyszukiwanie Podobieństwa:** Porównanie wektora zapytania z wektorami w bazie ChromaDB przy użyciu odległości cosinusowej. System zwraca k najbardziej podobnych fragmentów tekstu spełniających nałożone filtry.
4. **Montaż Kontekstu i Kontrola Limitów:** Uporządkowanie odnalezionych fragmentów według stopnia podobieństwa i połączenie ich w jeden blok kontekstowy. Kontrola liczby tokenów zapobiega przekroczeniu okna kontekstowego modelu LLM.
5. **Konstruowanie Promptu:** Budowa ostatecznego pytania. Prompt zawiera instrukcje systemowe (np. zakaz halucynowania), kontekst z przypisami oraz treść pytania użytkownika.
6. **Generowanie Odpowiedzi:** Przekazanie promptu do modelu językowego (np. GPT-5.4 lub Llama 3.2 3B) drogą sieciową lub lokalnie. Strumieniowe przesyłanie tokenów (ang. *streaming*) skraca czas oczekiwania na pierwszą część odpowiedzi.
7. **Postprocessing i Prezentacja Wyników:** Weryfikacja składni i zmapowanie przypisów na rzeczywiste adresy URL. Gotowy tekst trafia do interfejsu użytkownika.

3.2.4 Diagram przepływu danych

Aby czytelnie zobrazować i zestawić ze sobą procesy zachodzące w czasie indeksowania oraz wyszukiwania, opracowano precyzyjny schemat przepływu danych. Rysunek 3.2 przedstawia podział na asynchroniczny potok indeksowania dokumentów oraz synchroniczny potok przetwarzania zapytań i generowania odpowiedzi.



Rysunek 3.2: Przepływy przetwarzania danych w proponowanym systemie.

3.3 Metodologia przetwarzania danych

Jakość syntezy w systemach RAG zależy bezpośrednio od danych wejściowych. Obowiązuje tu reguła „śmieci na wejściu, śmieci na wyjściu” (ang. *garbage in, garbage out*). W dalszej części rozdziału przedstawiono zbiór testowy, sposoby podziału tekstu, strukturę metadanych oraz procesy normalizacji.

3.3.1 Charakterystyka zbioru testowego

Ewaluację systemu przeprowadzono na zdywersyfikowanym zbiorze artykułów zgromadzonym w pliku `evaluation_dataset.json`. Treści te obejmują doniesienia prasowe, opracowania popularnonaukowe, recenzje oraz publicystykę społeczno-gospodarczą. Wykorzystano publikacje z portali Reuters, BBC, TechCrunch i Ars Technica. Rozpiętość dokumentów waha się od 500 do 10 000 słów. Pozwala to na zbadanie stabilności algorytmów podziału tekstu przy zróżnicowanej objętości kontekstu. Dla każdego artykułu przygotowano trudne pytania testowe (łącznie 25 pozycji dla całego zbioru). Obejmują one zagadnienia faktograficzne, porównawcze oraz wymagające syntezy wiedzy.

3.3.2 Strategie dzielenia tekstu na fragmenty

Sposób podziału tekstu decyduje o spójności semantycznej danych wejściowych w bazie wektorowej. Porównano trzy podejścia. Pierwsze to podział o stałym rozmiarze (ang. *Fixed-Size Chunking*) na bloki o określonej liczbie tokenów (np. 500). Prostota wdrożenia wiąże się tu jednak z ryzykiem rozrywania zdań, co psuje jakość osadzeń. Drugie podejście, czyli podział akapitowy (ang. *Paragraph-Based Chunking*), opiera się na znacznikach akapitów. Gwarantuje to spójność myśli autora, lecz generuje fragmenty o skrajnie różnej długości. Trzecie, wybrane w tym projekcie rozwiązanie, to hybrydowy podział zdaniowy (ang. *Hybrid Sentence-Based Chunking*). Analizator językowy dzieli tekst na zdania, które są łączone aż do osiągnięcia zadanego limitu tokenów. Nowy blok zachowuje jedno- lub dwuzdaniowe nakładanie z poprzedniego. Chroni to strukturę składniową i spójność wyводу. Wpływ długości fragmentu zbadano empirycznie dla rozmiarów: 250 (konfiguracja C5), 500 (C1) oraz 1000 tokenów (C6).

3.3.3 Ekstrakcja metadanych

Każdy rekord w bazie ChromaDB zawiera metadane strukturalne. Zapobiegają one krzyżowemu zanieczyszczeniu kontekstu podczas odpytywania bazy i pozwalają ograniczyć przeszukiwanie do konkretnego artykułu. Metadane ułatwiają też mapowanie źródeł oraz odtworzenie pierwotnej struktury tekstu. Tabela 3.1 zawiera schemat metadanych stosowanych w systemie.

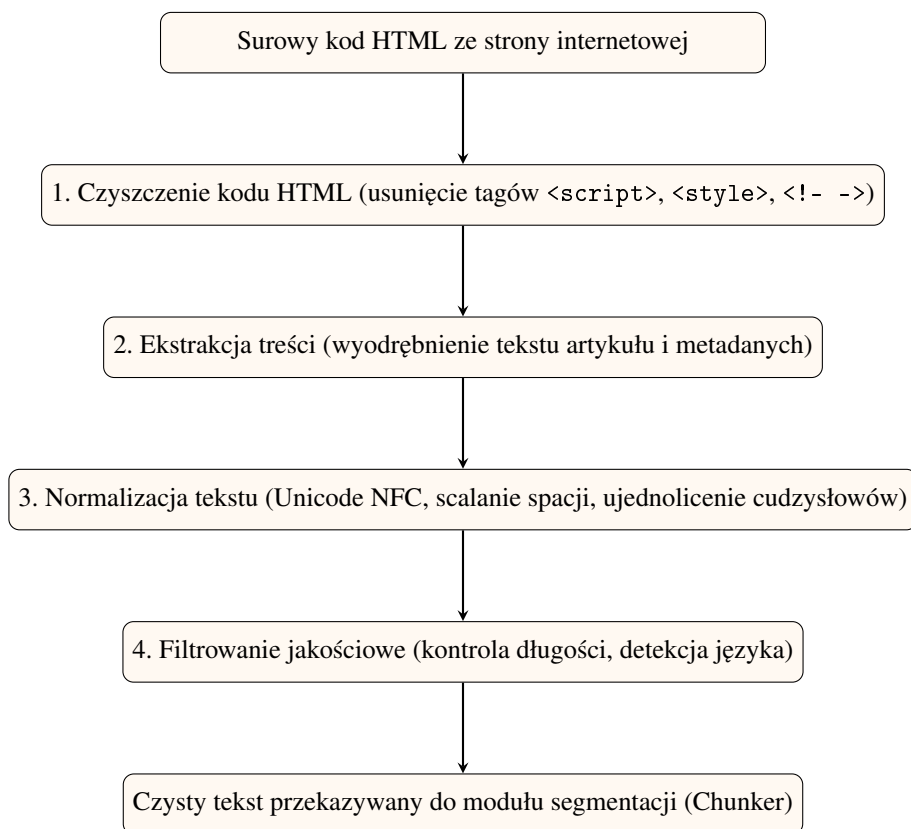
Tabela 3.1: Schemat struktury metadanych dla fragmentów tekstowych w bazie wektorowej.

Klucz pola	Typ danych	Wymagalność	Rola i przeznaczenie w systemie
source_url	string	Tak	Identyfikacja oryginalnego adresu URL artykułu.
title	string	Tak	Tytuł artykułu wykorzystywany w cytowaniach.
chunk_index	integer	Tak	Pozycja fragmentu w dokumencie (0-indexed).
total_chunks	integer	Tak	Całkowita liczba fragmentów danego dokumentu.
author	string	Nie	Informacja o autorze tekstu źródłowego.
publish_date	string	Nie	Data publikacji artykułu do analizy chronologicznej.
word_count	integer	Nie	Liczba słów w danym fragmencie tekstowym.

3.3.4 Potok przetwarzania wstępnego

Przed wektoryzacją dokument przechodzi przez potok przygotowawczy. Pozwala to odrzucić znaki sterujące, kod skryptów czy puste akapity. Etapy czyszczenia oraz normalizacji tekstu ilustruje Rysunek 3.3.

Filtry jakościowe odrzucają bloki tekstu o niskiej liczbie unikalnych słów. Chroni to system przed zapisywaniem błędnie pobranych witryn. Odrzucane są również artykuły w języku niezgodnym z wybranym dla danej sesji.



Rysunek 3.3: Etapy przetwarzania wstępnego w wejściowym potoku danych.

3.4 Metodologia ewaluacji

Rygorystyczna ewaluacja empiryczna stanowi podstawę naukowej walidacji systemu. W rozdziale określono wymiary oceny, zdefiniowano matematycznie metryki wyszukiwania oraz opisano kryteria pomiaru jakości generowania odpowiedzi i wydajności obliczeniowej. Szczegółowy plan eksperymentów obejmuje osiem odmiennych konfiguracji testowych.

3.4.1 Wymiary ewaluacji

Ewaluację systemu przeprowadzono w czterech wymiarach. Pierwszym jest jakość wyszukiwania (ang. *Retrieval Quality*). Mierzy ona zdolność retrievera do odnajdywania fragmentów tekstu zawierających odpowiedzi na pytania. Drugi wymiar stanowi jakość generowania (ang. *Generation Quality*). Ocenie poddano tu wierność źródłom, spójność logiczną oraz brak halucynacji w odpowiedziach modelu LLM. Trzecim kryterium jest kompleksowa wydajność całego zintegrowanego potoku (end-to-end). Ostatni wymiar to efektywność obliczeniowa, mierzona czasem wykonania poszczególnych etapów i zużyciem zasobów.

3.4.2 Metryki wyszukiwania

Rygorystyczną ocenę modułu wyszukiwania semantycznego oparto na czterech klasycznych metrykach wyszukiwania informacji (ang. *Information Retrieval*, IR):

- **Precyzja dla k dokumentów (Precision@ k):** Stosunek liczby istotnych fragmentów wśród pierw-

szych k zwróconych wyników do całkowitej liczby k pobranych bloków:

$$\text{Precision@k} = \frac{|R_k \cap D_{rel}|}{k} \quad (3.1)$$

gdzie R_k to zbiór k najwyżej ocenionych fragmentów, a D_{rel} to zbiór dokumentów istotnych.

- **Czułość dla k dokumentów (Recall@k):** Określa odsetek odnalezionych istotnych fragmentów w stosunku do wszystkich poprawnych dokumentów w bazie referencyjnej:

$$\text{Recall@k} = \frac{|R_k \cap D_{rel}|}{|D_{rel}|} \quad (3.2)$$

- **Średni Odwrotny Rang (MRR, *Mean Reciprocal Rank*):** Mierzy pozycję pierwszego istotnego dokumentu w rankingu. Ma to duże znaczenie w architekturze RAG, ponieważ modele LLM najefektowniej korzystają z informacji umieszczonych na początku kontekstu:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \quad (3.3)$$

gdzie $|Q|$ to liczba zapytań, a rank_i to pozycja pierwszego poprawnego rekordu. W przypadku braku istotnego dokumentu wartość składnika wynosi 0.

- **Znormalizowany Skumulowany Zysk Dyskontowany (nDCG@k, *Normalized Discounted Cumulative Gain*):** Mierzy zarówno obecność dokumentów, jak i ich rzędy w rankingu. Wprowadza logarytmiczną karę za umieszczanie istotnych danych na dalekich pozycjach:

$$\text{DCG@k} = \sum_{i=1}^k \frac{2^{rel_i} - 1}{\log_2(i + 1)} \quad (3.4)$$

$$\text{nDCG@k} = \frac{\text{DCG@k}}{\text{IDCG@k}} \quad (3.5)$$

gdzie rel_i to stopień istotności dokumentu, a IDCG@k to idealny skumulowany zysk dyskontowany (ang. *Ideal DCG*) przy optymalnym rankingu.

3.4.3 Metryki generowania odpowiedzi

Ocena syntezy generowanej przez model językowy opiera się na czterech kryteriach. Pierwsze to wierność źródłom (ang. *Faithfulness*). Pozwala ono wykryć halucynacje poprzez weryfikację pokrycia tez w tekście źródłowym. Drugim kryterium jest adekwatność (ang. *Answer Relevance*). Sprawdza ono, czy model precyzyjnie odpowiada na intencję pytania, eliminując zbędne dygresje. Kolejne to spójność językowa (ang. *Coherence*), badająca płynność gramatyczną i logiczną struktury wypowiedzi. Ostatnie kryterium, kompletność (ang. *Completeness*), weryfikuje wyczerpanie wątków, co jest przydatne w pytaniach o charakterze porównawczym.

Ewaluacja ma charakter hybrydowy. Łączy oceny sędziów ludzkich (ankieta w skali Likerta 1–5) z oceną automatyczną. W tej drugiej zastosowano podejście „LLM jako sędzia” (ang. *LLM-as-a-judge*) [46]. Wtedy zewnętrzny, niezależny model językowy automatycznie ocenia pary prompt-odpowiedź według sformułowanych kryteriów.

3.4.4 Metryki wydajności

Wydajność operacyjna bezpośrednio wpływa na komfort użytkowania systemu. Analizie poddano następujące czasy opóźnień:

1. **Czas osadzania zapytania (ang. *Query Embedding Latency*):** czas wektoryzacji pytania przez model Sentence-Transformers.
2. **Czas przeszukiwania bazy (ang. *Vector Search Latency*):** czas wyszukiwania w bazie ChromaDB.
3. **Czas generowania odpowiedzi LLM (ang. *LLM Generation Latency*):** czas od wysłania promptu do otrzymania pełnej odpowiedzi lub pierwszego tokenu (ang. *Time to First Token*).
4. **Całkowity czas odpowiedzi (ang. *End-to-End Latency*):** sumaryczne opóźnienie systemu.

Mierzono także zużycie pamięci RAM oraz obciążenie CPU/GPU podczas indeksowania i odpytywania bazy.

3.4.5 Plan eksperymentów

Do systematycznego badania parametrów potoku RAG służy skrypt `run_comparative_tests.py`. Zaimplementowano w nim osiem konfiguracji eksperymentalnych (C1–C8). Każda z nich łączy wybrany model językowy, model osadzeń oraz rozmiar fragmentu tekstu. Szczegółowe parametry tych konfiguracji przedstawia Tabela 3.2.

Tabela 3.2: Szczegółowa specyfikacja konfiguracji badawczych (C1–C8) zdefiniowanych w module testowym.

Konfig.	Model LLM	Model osadzeń	Rozmiar chunk	Overlap	Backend LLM
C1	gpt-5.4	all-MiniLM-L6-v2	500 tokenów	50 tokenów	OpenAI API
C2	gpt-5.5	all-MiniLM-L6-v2	500 tokenów	50 tokenów	OpenAI API
C3	llama3.2 (3B)	all-MiniLM-L6-v2	500 tokenów	50 tokenów	Ollama (Lokalny)
C4	gpt-5.4	all-mpnet-base-v2	500 tokenów	50 tokenów	OpenAI API
C5	gpt-5.4	all-MiniLM-L6-v2	250 tokenów	25 tokenów	OpenAI API
C6	gpt-5.4	all-MiniLM-L6-v2	1000 tokenów	100 tokenów	OpenAI API
C7	claude-sonnet-4-6	Brak (Agentic)	N/A (Dynamiczny)	Brak	Anthropic API
C8	gpt-5-mini	all-MiniLM-L6-v2	500 tokenów	50 tokenów	OpenAI API

Struktura eksperymentów ułatwia izolację poszczególnych zmiennych. Bezpośrednie porównania obejmują:

- **Wpływ modelu językowego:** zestawienie konfiguracji C1 (lekki model gpt-5.4), C2 (flagowy model gpt-5.5) oraz C3 (lokalna Llama 3.2 3B).
- **Wpływ modelu osadzeń:** porównanie konfiguracji C1 (model o wymiarze 384) z konfiguracją C4 (model all-mpnet-base-v2 o wymiarze 768).
- **Wpływ rozmiaru fragmentu:** analiza konfiguracji C1 (500 tokenów), C5 (250 tokenów) oraz C6 (1000 tokenów), przy nakładaniu rzędu 10%.
- **Klasyczny RAG a podejście agentowe:** porównanie wariantu C1 (indeksowanie i baza wektorowa) z wariantem C7 (model claude-sonnet-4-6 wyszukujący dane dynamicznie w czasie zapytania). Pomaga to ocenić kompromis między czasem a jakością.

- **Optymalizacja kosztów chmury:** porównanie konfiguracji **C3** (lokalna Llama 3.2 3B) z konfiguracją **C8** (tani model gpt-5-mini), co pozwala ocenić opłacalność chmurowych wersji budżetowych.

Wszelkie testy są uruchamiane automatycznie w sposób izolowany na tym samym zestawie danych wejściowych, a wyniki są zapisywane w formacie JSON w dedykowanych podkatalogach katalogu `experiments/`.

3.4.6 Tworzenie danych referencyjnych

W celu zapewnienia rzetelności ewaluacji dane referencyjne (tzw. *ground truth*) przygotowano ręcznie. Proces ten przebiegał według określonych wytycznych. Najpierw wybrano 5 reprezentatywnych artykułów ze zbioru testowego, które tworzyły tzw. zbiór ślepy, nieużywany podczas strojenia promptów. Dla każdej publikacji sformułowano po 5 pytań o różnym stopniu trudności, od prostego odpytywania o fakty po syntezę wątków. Następnie oznaczono fragmenty tekstu źródłowego będące uzasadnieniem merytorycznym. Na tej podstawie zredagowano precyzyjne odpowiedzi referencyjne (ang. *gold standard answers*). Tak przygotowana baza pozwala ocenić precyzję wyszukiwania i merytoryczność syntezy modeli LLM.

3.4.7 Analiza statystyczna

Zgromadzone wyniki ilościowe poddano analizie statystycznej. Pozwala to ocenić, czy różnice między konfiguracjami (C1–C8) wynikają z przewag technologicznych, czy stanowią wynik losowych wahań. Istotność statystyczną weryfikuje się parowanym testem t-Studenta [47] lub nieparametrycznym testem Wilcoxa [48], zależnie od rozkładu opóźnień. Przedziały ufności na poziomie 95% wyznacza metoda bootstrapowa dla uśrednionych metryk nDCG@k oraz Recall@k. Do oceny praktycznego znaczenia różnic służy współczynnik *d* Cohena określający wielkość efektu. Taka analiza ułatwia sformułowanie wniosków dotyczących doboru komponentów potoku RAG.

3.5 Kwestie etyczne

Automatyczne pobieranie treści z sieci i stosowanie modeli generatywnych wiąże się z wyzwaniami etycznymi i prawnymi. Analizie poddano zagadnienia ochrony prywatności, prawa autorskie, stronniczość algorytmów oraz wpływ energetyczny proponowanego systemu.

3.5.1 Prywatność danych i ochrona informacji

System przetwarza wyłącznie dane publiczne. Projekt nie zawiera modułów do omijania zapór (paywalli) czy paneli logowania. Rozwiązanie spełnia wymagania rozporządzenia o ochronie danych osobowych (RODO). Pierwszą zasadą jest minimalizacja danych, ponieważ w bazie ChromaDB zapisywane są tylko wektory i fragmenty tekstów niezbędne do syntezy odpowiedzi. Nie gromadzi się profili ani historii zapytań. Drugą zasadę stanowi prawo do bycia zapomnianym. ChromaDB pozwala na natychmiastowe usunięcie wszystkich fragmentów powiązanych z określonym adresem URL na żądanie właściciela witryny.

3.5.2 Prawa autorskie i dozwolony użytek

Analiza semantyczna pobranych tekstów mieści się w granicach dozwolonego użytku naukowo-badawczego. Respektowanie praw autorskich twórców zapewniają trzy wbudowane mechanizmy. Pierwszym jest obowiązkowa atrybucja – każda wygenerowana odpowiedź jest uzupełniana o odsyłacze do źródłowych adresów URL. Drugi mechanizm to blokada redystrybucji. Aplikacja nie udostępnia

całych dokumentów, a jedynie generuje odpowiedzi, pokazując krótkie, wyrwane z kontekstu cytaty do weryfikacji. Ostatnim elementem jest poszanowanie dyrektyw w plikach `robots.txt`. System pomija witryny, których właściciele zablokowali automatyczne skanowanie.

3.5.3 Stronniczość modeli i wiarygodność informacji

Modele językowe i algorytmy osadzeń mogą powielać uprzedzenia obecne w danych treningowych [49, 50]. Pobieranie tekstów z sieci niesie też ryzyko zaindeksowania dezinformacji. Do ograniczenia tych zjawisk zastosowano dwa rozwiązania. Pierwszym są instrukcje systemowe wymuszające neutralność i zakazujące spekulacji wykraczających poza dostarczony kontekst. Drugie to mechanizm informowania o braku odpowiedzi. Gdy kontekst nie zawiera faktów potrzebnych do sformułowania odpowiedzi, model generuje formułę: „Brak wystarczających informacji w dostarczonych źródłach”. Zapobiega to halucynacji.

3.5.4 Wpływ środowiskowy

Praca modeli głębokiego uczenia wiąże się z dużym zużyciem energii i śladem węglowym. W projekcie ograniczono to zużycie na trzy sposoby. Po pierwsze, wybrano zdestylowany model osadzeń `all-MiniLM-L6-v2`, oszczędny pod względem zapotrzebowania na moc obliczeniową. Po drugie, zbadano model `Llama 3.2` (3 miliardy parametrów) uruchamiany lokalnie. Pozwala on na generowanie odpowiedzi na standardowych stacjach roboczych bez obciążania serwerów w centrach danych. Po trzecie, wdrożono buforowanie (ang. *caching*) zapytań wektorowych i generatywnych, co eliminuje powtarzanie tych samych operacji obliczeniowych.

3.6 Ograniczenia aplikacyjne

Określenie barier technicznych i merytorycznych pozwala na bezpieczne i efektywne korzystanie z systemu. Opisano tu zakres funkcjonalny prototypu oraz jego ograniczenia sprzętowe i jakościowe.

3.6.1 Zakres rozwiązania

Prototyp systemu analizy treści internetowych ma określone granice użytkowe. Przetwarza wyłącznie czysty tekst wyodrębniony ze stron internetowych, pomijając obrazy, wideo, nagrania audio oraz wykresy. Badania skupiły się na języku angielskim, co odzwierciedla specyfikę globalnej sieci. System działa w trybie „na żądanie” (ang. *on-demand*) i nie wspiera ciągłego monitorowania zmian na stronach (ang. *continuous web scraping*) ani automatycznej aktualizacji indeksów.

3.6.2 Ograniczenia techniczne i jakościowe

Zidentyfikowano cztery główne ograniczenia inżynierskie systemu. Pierwszym są zależności sieciowe. Komercyjne konfiguracje (C1, C2, C4, C5, C6, C7, C8) wymagają stabilnego łącza z serwerami OpenAI lub Anthropic, a opóźnienia sieciowe i limity zapytań bezpośrednio wpływają na czas reakcji. Drugie to wymagania sprzętowe trybu lokalnego. Uruchomienie konfiguracji **C3** z modelem `Llama 3.2 (3B)` wymaga procesora wielordzeniowego i minimum 8 GB pamięci RAM. Bez akceleracji GPU czas syntezy odpowiedzi rośnie z 2 do ponad 15 sekund. Trzecie ograniczenie to spadek precyzji dla długich dokumentów (powyżej 50 000 słów). Proste wyszukiwanie wektorowe bez algorytmu rerankingu gubi wtedy istotny kontekst (zjawisko „zagubienia w środku”, ang. *lost in the middle*). Ostatnim problemem jest wrażliwość na jakość kodu HTML. Witryny o niestandardowej strukturze lub zabezpieczone przed botami mogą uniemożliwić ekstrakcję tekstu.

3.7 Podsumowanie rozdziału

Rozdział ten przedstawia metodologię i założenia systemu semantycznej analizy treści internetowych opartego na technologii RAG. Zdefiniowano w nim cztery pytania badawcze i osadzono prace w ramach badawczych Design Science Research (DSR). Modułowa architektura dzieli procesy na asynchroniczne indeksowanie oraz synchroniczną obsługę zapytań, co ilustrują schematy przepływu danych.

Zaprezentowano schemat metadanych i hybrydową segmentację zdaniową. Opracowano plan eksperymentów obejmujący osiem konfiguracji (C1–C8), badających wpływ poszczególnych komponentów na jakość wyszukiwania i generowania odpowiedzi. Całość dopełnia omówienie kwestii etycznych oraz ograniczeń technicznych systemu. Przedstawiony projekt i ramy badawcze stanowią podbudowę dla kolejnych rozdziałów, opisujących zastosowane technologie oraz szczegóły implementacyjne.

Rozdział 4

Zastosowane technologie i narzędzia

Niniejszy rozdział opisuje technologie, biblioteki oraz narzędzia wykorzystane podczas budowy prototypu, w oderwaniu od szczegółów samej implementacji. Wybór poszczególnych komponentów oparto na dojrzałości ekosystemu, wydajności oraz zgodności z otwartym oprogramowaniem, ponieważ kryteria te decydują o stabilności całego systemu. W kolejnych podrozdziałach przedstawiono język programowania, framework webowy, model osadzeń, bazę wektorową, modele językowe oraz biblioteki pomocnicze.

4.1 Język programowania — Python

Jako fundament aplikacyjny wybrano język Python (wersja 3.10 lub nowsza). Stanowi on standard w dziedzinie przetwarzania języka naturalnego (NLP). Dojrzałość bibliotek RAG (takich jak Sentence-Transformers) oraz szybkość prototypowania zdecydowały o tym wyborze. Praca z kodem jest dzięki temu efektywna.

Niższa wydajność Pythona względem języków kompilowanych, takich jak Rust czy C++, nie stanowi bariery w tym projekcie. Kluczowe, najbardziej obciążające procesor operacje obliczeniowe (generowanie osadzeń i inferencja LLM) wykonują zoptymalizowane silniki napisane w C/C++ (np. llama.cpp, PyTorch) lub dedykowane serwery chmurowe. Bezpieczeństwo typów zapewniają adnotacje typów (ang. *type hints*) oraz walidacja struktur danych za pomocą biblioteki Pydantic [51].

4.2 Framework RAG — Haystack

Orkiestrację komponentów potoku RAG powierzono bibliotece Haystack v2 [52], rozwijanej przez firmę deepset. Framework ten organizuje przetwarzanie danych w postaci grafu skierowanego (potoku), w którym poszczególne komponenty (konwertery dokumentów, embeddery, generatory odpowiedzi) łączy się deklaratywnie w sekwencję operacji. Ułatwia to wymianę pojedynczego elementu potoku (np. modelu osadzeń czy generatora LLM) bez ingerencji w pozostałą część kodu. Alternatywą dla tego rozwiązania są biblioteki LangChain oraz LlamaIndex, jednak Haystack wybrano ze względu na jego modułną architekturę komponentową oraz natywne wsparcie integracji z bazą ChromaDB.

4.3 Framework webowy — FastAPI

Do udostępnienia funkcji systemu przez interfejs API wybrano framework FastAPI [53]. Oferuje on wysoką wydajność, opartą na technologii Starlette i bibliotece Pydantic. Pełna obsługa asynchroniczności (mechanizm `async/await`) ułatwia wykonywanie nieblokujących operacji wejścia-wyjścia (I/O). Atutem jest też automatyczne generowanie dokumentacji OpenAPI. Tabela 4.1 przedstawia porównanie FastAPI z alternatywnymi rozwiązaniami w ekosystemie Pythona.

Flask sprawdza się w mikroprojektach, lecz brak walidacji danych i konieczność ręcznego opisywania API wydłużyłyby prace wdrożeniowe. Z kolei Django to system zbyt rozbudowany. Posiada własny ORM, system szablonów i panel administratora, co generuje narzut pamięciowy i utrudnia konteneryzację. FastAPI łączy lekkość mikroframeworka z nowoczesnymi mechanizmami walidacji.

Tabela 4.1: Porównanie frameworków webowych w języku Python pod kątem przydatności w systemach RAG.

Cecha / Parametr	FastAPI	Flask	Django
Natywna asynchroniczność	Tak	Ograniczona (rozszerzenia)	Ograniczona
Automatyczna dokumentacja API	Tak (OpenAPI)	Nie	Nie
Wydajność wykonawcza	Bardzo wysoka	Średnia	Średnia
Automatyczna walidacja typów	Tak (Pydantic)	Nie	Nie
Krzywa uczenia się	Niska	Niska	Średnia/Wysoka
Złożoność konfiguracji	Niska	Niska	Wysoka

4.4 Wybór modelu osadzeń

Zdolność systemu do rozumienia tekstu zależy bezpośrednio od wybranego modelu osadzeń. Jako bazowy komponent wybrano all-MiniLM-L6-v2 z biblioteki Sentence-Transformers [27]. Ta zdestylowana, zoptymalizowana wersja modelu BERT charakteryzuje się dobrym stosunkiem jakości reprezentacji do narzutu obliczeniowego. Tabela 4.2 zawiera zestawienie modeli branych pod uwagę przy projektowaniu.

Tabela 4.2: Charakterystyka porównawcza wybranych modeli osadzeń tekstowych.

Nazwa modelu	Wymiary	Max tokenów	Rozmiar na dysku	Typ dostępu
all-MiniLM-L6-v2	384	256	~80 MB	Lokalny (wolny)
all-mpnet-base-v2	768	384	~420 MB	Lokalny (wolny)
BGE-large-en-v1.5	1024	512	~1.34 GB	Lokalny (wolny)
text-embedding-3-small	1536	8191	Brak (chmura)	Płatne API
text-embedding-3-large	3072	8191	Brak (chmura)	Płatne API

Model all-MiniLM-L6-v2 można uruchomić na komputerach bez dedykowanej karty graficznej (GPU). Na standardowym procesorze CPU szybkość generowania osadzeń przekracza 14 000 zdań na sekundę. W celu porównania i obsługi bardziej wymagających zadań system zintegrowano z modelem all-mpnet-base-v2. Generuje on wektory o wymiarze 768. Ułatwia to wychwytywanie subtelnych zależności semantycznych, lecz wymaga więcej czasu i pamięci.

4.5 Baza wektorowa — ChromaDB

Do zapisywania i wyszukiwania wektorów wybrano osadzoną (ang. *embedded*) bazę danych ChromaDB [29]. Działa ona w procesie aplikacji FastAPI, co eliminuje sieciowy narzut komunikacyjny. Brak konieczności konfiguracji klastrów ułatwia prowadzenie badań i uruchamianie systemu. Całość bazuje na SQLite, zapisując indeksy i metadane w plikach lokalnych, co upraszcza tworzenie kopii zapasowych.

Biblioteka FAISS [34] cechuje się wysoką wydajnością przy masowych zbiorach danych. Jest to jednak rozwiązanie niskopoziomowe, pozbawione wbudowanej obsługi metadanych i mechanizmów filtrowania. Narzędzia takie jak Milvus czy Weaviate wymagają uruchamiania klastrów kontenerów za pomocą Docker Compose, co zwiększa złożoność administracyjną. ChromaDB stanowi kompromis. Zapewnia filtrowanie, persystencję i zarządzanie metadanymi bezpośrednio w pamięci aplikacji.

Tabela 4.3: Porównanie systemów zarządzania bazami wektorowymi.

Baza danych	Typ wdrożenia	Skalowalność	Złożoność	Koszt	Metadane
ChromaDB	Osadzona / Serwer	Średnia	Bardzo niska	Darmowa	Pełne wsparcie
Pinecone	Chmura (SaaS)	Bardzo wysoka	Niska	Płatna	Pełne wsparcie
Weaviate	Serwer / Klaster	Wysoka	Średnia	Darmowa	Pełne wsparcie
Milvus	Rozproszona	Ekstremalna	Wysoka	Darmowa	Pełne wsparcie
FAISS	Biblioteka	Wysoka	Średnia	Darmowa	Brak (wymaga kodu)

4.6 Wybór modelu językowego

Generację odpowiedzi powierzono dużym modelom językowym. Aby rozszerzyć zakres zastosowań systemu, zintegrowano w nim komercyjne modele chmurowe oraz otwarte modele lokalne. Konfiguracja chmurowa wykorzystuje modele OpenAI (GPT-5.4 oraz GPT-5.5) [3]. Wariant gpt-5.4 wyróżnia się krótkim czasem generowania, niskim kosztem eksploatacji oraz oknem kontekstowym rzędu 128 000 tokenów. Z kolei flagowy model gpt-5.5 reprezentuje punkt odniesienia w zakresie zaawansowanego wnioskowania logicznego i planowania. Do pracy lokalnej wdrożono model Llama 3.2 [21] w wariacie 3B (3 miliardy parametrów) z domyślną kwantyzacją Ollama Q4_K_M, uruchamiany za pośrednictwem środowiska Ollama [54]. Pozwala on na darmowe przetwarzanie informacji w trybie offline i gwarantuje poufność danych. Jego rozmiar umożliwia płynną pracę na typowych procesorach CPU oraz GPU, z zachowaniem poprawności gramatycznej w języku polskim oraz angielskim. Taka hybrydowa architektura zabezpiecza system przed przerwami w działaniu usług zewnętrznych i pozwala optymalizować koszty.

Do ujednolicenia komunikacji z różnymi dostawcami API wykorzystano bibliotekę LiteLLM [55], która działa jako warstwa pośrednicząca (proxy) o jednolitym interfejsie programistycznym. Dzięki temu zmiana dostawcy modelu (OpenAI, lokalny Ollama) sprowadza się do modyfikacji parametru konfiguracyjnego, bez konieczności zmiany kodu wywołującego generator.

4.7 Biblioteki do ekstrakcji treści ze stron

Pobieranie i parsowanie stron internetowych oparto na dwóch komplementarnych technologiach. Do pobierania statycznych witryn służy połączenie bibliotek requests [56] oraz BeautifulSoup4 [57]. Umożliwia ono szybkie parsowanie drzewa DOM (ang. *Document Object Model*). Z kolei dynamiczne witryny typu SPA (ang. *Single Page Application*), budowane w oparciu o frameworki React, Angular czy Vue, obsługuje silnik Playwright [58]. Uruchamia on bezgłową (ang. *headless*) przeglądarkę Chromium, wykonuje skrypty JavaScript i pobiera wyrenderowany tekst. Hybrydowe podejście chroni przed pobraniem pustej zawartości (ang. *empty body*), co jest częstym błędem prostych parserów.

4.8 Dodatkowe biblioteki

Uzupełnieniem architektury są biblioteki pomocnicze zapewniające stabilność operacji na strukturach danych. Ich zestawienie oraz zastosowanie zawiera Tabela 4.4.

Tabela 4.4: Zależności programistyczne i biblioteki pomocnicze systemu.

Biblioteka	Minimalna wersja	Główny cel i zastosowanie w systemie
pydantic [51]	2.0	Walidacja typów danych wejściowych i wyjściowych API.
python-dotenv [59]	1.0	Bezpieczne zarządzanie kluczami API i zmiennymi środowiskowymi.
tiktoken [60]	0.5	Dokładne szacowanie liczby tokenów dla promptów OpenAI.
numpy [61]	1.24	Operacje matematyczne na wektorach i obliczenia podobieństwa.

Rozdział 5

Implementacja systemu

Rozdział opisuje techniczną implementację zaprojektowanego systemu. Architektura opiera się na paradygmacie RAG (ang. *Retrieval-Augmented Generation*), łączącym wyszukiwanie informacji z modelami generatywnymi. Przedstawiono stos technologiczny, architekturę oprogramowania oraz integrację poszczególnych komponentów. Osobne sekcje poświęcono potokom indeksowania i zapytań, warstwie API oraz konteneryzacji środowiska.

5.1 Wprowadzenie i stos technologiczny

Do stworzenia systemu wybrano język Python (wersja 3.11). Decyzję tę uzasadnia dojrzałość tutejszego ekosystemu w obszarach uczenia maszynowego i inżynierii danych. Dobór pozostałych bibliotek wynikał z wymogów wydajnościowych oraz konieczności zachowania pełnej modularności kodu. Kluczowe znaczenie miała też sprawna praca z wektorami.

Tabela 5.1 zestawia najważniejsze komponenty stosu technologicznego. Szczegółowe uzasadnienie wyboru wymienionych technologii zawiera rozdział 4.

Tabela 5.1: Stos technologiczny zaimplementowanego systemu analizy treści internetowych.

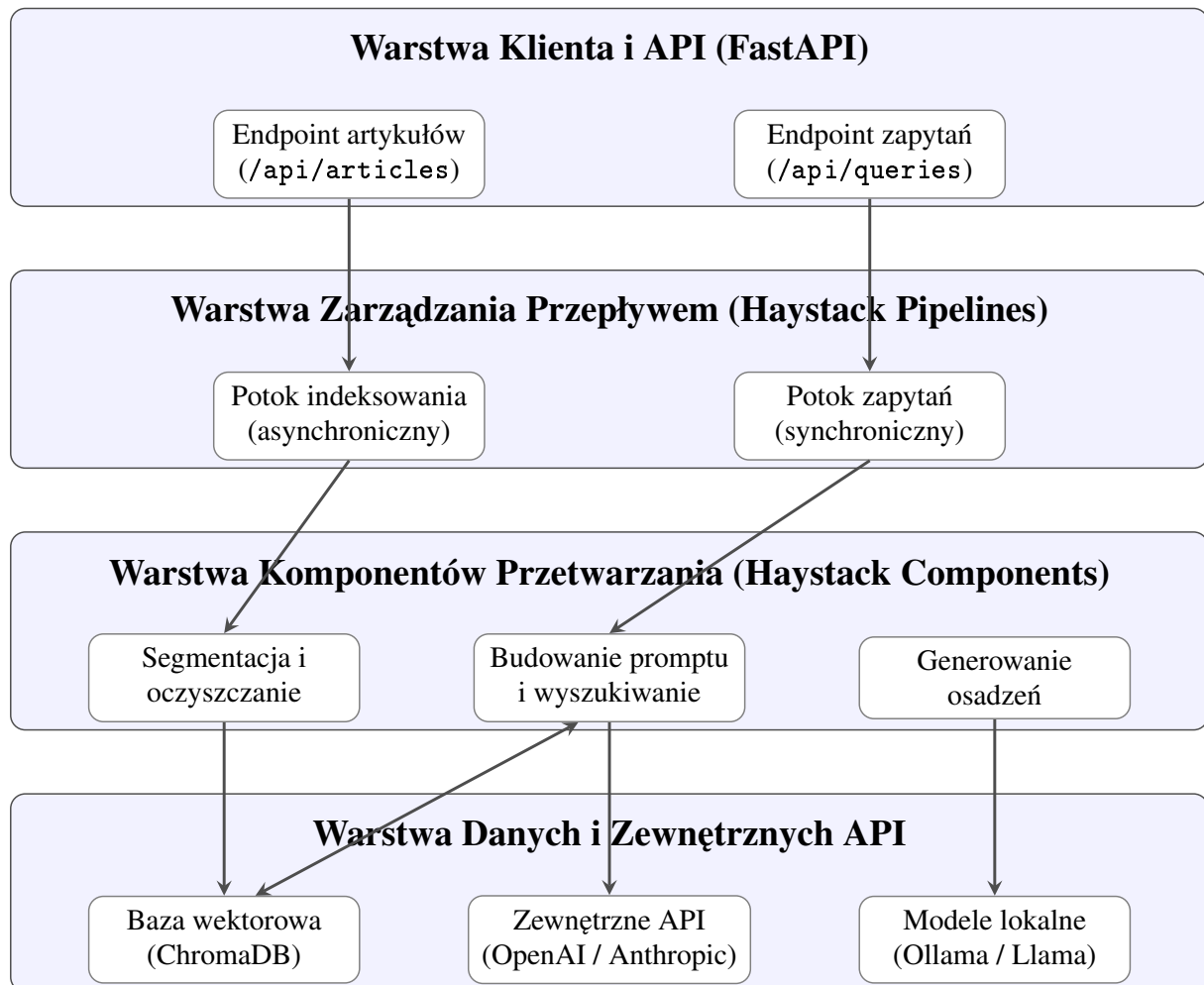
Komponent	Technologia	Uzasadnienie doboru
Język programowania	Python 3.11	Bogaty ekosystem, dojrzałość bibliotek ML/NLP, wysoka produktywność.
Framework RAG	Haystack v2	Architektura oparta na komponentach i potokach (DAG), wysoka modularność.
Baza wektorowa	ChromaDB	Praca w trybie wbudowanym (<i>embedded</i>), natywne filtrowanie metadanych, brak narzutu administracyjnego.
Framework API	FastAPI	Wysoka wydajność, asynchroniczność (async/await), automatyczna dokumentacja OpenAPI/Swagger.
Integracja LLM	LiteLLM / Ollama	Jednolity interfejs dla zewnętrznych modeli chmurowych (OpenAI) oraz modeli lokalnych.
Scraping treści	BS4 / Playwright	BS4 do szybkich stron statycznych, Playwright do dynamicznych stron SPA.

5.2 Architektura systemu i przepływ danych

System oparto na architekturze warstwowej. Wydzielono w niej interfejs API, logikę aplikacyjną (potoki i usługi) oraz warstwę dostępu do danych, w tym bazy wektorowe i zewnętrzne API. Do integracji poszczególnych modułów wykorzystano klasyczne wzorce projektowe. Są to przede wszystkim fabryka (ang. *Factory Pattern*) oraz wstrzykiwanie zależności (ang. *Dependency Injection*).

Rysunek 5.1 obrazuje ten podział wraz z kierunkami przepływu informacji.

Przetwarzanie danych w systemie realizują dwa niezależne procesy:



Rysunek 5.1: Schemat architektury warstwowej zaimplementowanego systemu analizy treści.

1. **Potok indeksowania** (asynchroniczny) – odpowiada za pobieranie treści z podanego adresu URL, usuwanie kodu HTML wraz z elementami nawigacyjnymi oraz podział tekstu na fragmenty semantyczne. Dla każdego z nich generowany jest wektor osadzenia, po czym dane trafiają do bazy ChromaDB.
2. **Potok zapytań** (synchroniczny) – przekształca pytanie użytkownika w wektor osadzenia, pobiera z bazy ChromaDB najbardziej dopasowane fragmenty i na ich podstawie konstruuje ostateczny prompt. Odpowiedź generowana przez model zawiera bezpośrednie odniesienia do materiałów źródłowych.

5.3 Implementacja głównych komponentów

Działanie systemu opiera się na kilku komponentach odpowiedzialnych za konfigurację, tworzenie generatorów tekstu oraz przygotowanie danych dla bazy wektorowej. W kolejnych podrozdziałach opisano moduł zarządzania ustawieniami, fabrykę generatorów oraz autorski komponent oczyszczający metadane na potrzeby bazy ChromaDB.

5.3.1 Moduł konfiguracji systemu i zarządzania ustawieniami

Do zarządzania konfiguracją wykorzystano bibliotekę pydantic-settings. Ułatwia ona deklaratywne definiowanie zmiennych środowiskowych. Za ich automatyczne rzutowanie oraz odczyt z pliku `.env` odpowiada klasa `Settings` z modułu `app_haystack/config.py`. Zaimplementowano tu także mechanizm nadpisywania parametrów w locie za pomocą pliku `settings_override.json`. Uproszczoną definicję klasy `Settings` wraz z metodą budującą konfigurację generatora przedstawia Listing 5.1.

Listing 5.1: Deklaratywna konfiguracja systemu oparta na bibliotece pydantic-settings (fragment klasy `Settings`).

```
1 from typing import Literal, Optional
2 from pydantic_settings import BaseSettings, SettingsConfigDict
3
4
5 class Settings(BaseSettings):
6     model_config = SettingsConfigDict(env_file=".env", extra="ignore")
7
8     llm_backend: Literal["openai", "ollama"] = "ollama"
9     openai_model: str = "gpt-5.4"
10    ollama_model: str = "llama3.2"
11    embedding_backend: Literal["sentence-transformers", "openai"] = "
        sentence-transformers"
12    embedding_model: str = "all-MiniLM-L6-v2"
13    chunk_size: int = 500
14    chunk_overlap: int = 50
15    retrieval_top_k: int = 5
16    anthropic_model: str = "claude-sonnet-4-6"
17
18    def get_llm_config(self) -> dict:
19        if self.llm_backend == "openai":
20            return {"backend": "openai",
21                    "model": self.llm_model or self.openai_model,
22                    "api_key": self.openai_api_key,
23                    "api_base_url": self.openai_base_url}
24        return {"backend": "ollama",
25                "model": self.llm_model or self.ollama_model,
```

```
26         "base_url": self.ollama_base_url}
27
28
29 settings = Settings()
30 settings.apply_overrides()
```

5.3.2 Fabryka generatorów i obsługa interfejsów API

Proces tworzenia generatorów tekstu ujednolicono w module fabryki (`app_haystack/factories/generator.py`). Narzędzie to analizuje słownik konfiguracyjny. Następnie powołuje do życia właściwy komponent Haystack – `OpenAIGenerator` bądź `OllamaChatGenerator`. Ważnym usprawnieniem jest obsługa parametru `api_base_url` w konfiguracji klienta OpenAI. Dzięki temu zapytania można łatwo przekierować na niestandardowe proxy, np. LiteLLM, co uprościło testowanie zewnętrznych modeli. Funkcję fabrykującą generator na podstawie słownika konfiguracyjnego prezentuje Listing 5.2.

Listing 5.2: Funkcja fabryki tworząca właściwy generator Haystack na podstawie konfiguracji backendu.

```
1 def create_generator(config: dict):
2     backend = config.get("backend", "openai")
3     model = config.get("model")
4     temperature = config.get("temperature", 0.1)
5     max_tokens = config.get("max_tokens", 1024)
6
7     if backend == "openai":
8         generation_kwargs = {"max_tokens": max_tokens}
9         # modele wnioskujące (np. gpt-5.5) wymagają temperatury równej
10        1.0
11        if model and any(x in model.lower() for x in ["gpt-5.5", "o1", "o3"]):
12            generation_kwargs["temperature"] = 1.0
13        else:
14            generation_kwargs["temperature"] = temperature
15        return OpenAIGenerator(
16            api_key=Secret.from_token(config["api_key"]),
17            model=model,
18            api_base_url=config.get("api_base_url"),
19            generation_kwargs=generation_kwargs,
20        )
21
22    elif backend == "ollama":
23        return OllamaChatGenerator(
24            model=model,
25            url=config.get("base_url"),
26            generation_kwargs={"temperature": temperature,
27                               "num_predict": max_tokens},
28        )
29
30    raise ValueError(f"Unknown LLM backend: {backend}")
```

5.3.3 Rozwiązanie problemu oczyszczania metadanych (MetadataCleaner)

Podczas integracji bazy ChromaDB z potokiem Haystack wykryto błąd. Silnik bazy cicho odrzucał dokumenty. Przyczyną okazał się format metadanych przesyłanych z komponentu `MetadataEnricher`.

Dodawał on do dokumentów listy, na przykład `tags = ["article_1"]`. Tymczasem ChromaDB akceptuje wyłącznie typy proste: liczby, wartości logiczne i napisy. Próba zapisu złożonych struktur kończyła się niepowodzeniem bez zgłaszania wyjątków w wyższych warstwach aplikacji. Skutkowało to całkowitym brakiem wyników w wyszukiwaniu semantycznym (`sources_retrieved = 0`).

Aby usunąć tę barierę, opracowano autorski komponent pośredniczący `MetadataCleaner` (`app_haystack/components/metadata_cleaner.py`). Jego kod źródłowy prezentuje listing 5.3.

Listing 5.3: Kod źródłowy niestandardowego komponentu `MetadataCleaner` oczyszczającego metadane dla ChromaDB.

```
1 import logging
2 from typing import List, Dict, Any
3 from haystack import component, Document
4
5 logger = logging.getLogger(__name__)
6
7 @component
8 class MetadataCleaner:
9     """Oczyszcza metadane dokumentów w celu zapewnienia kompatybilności
10        z ChromaDB."""
11
12     @component.output_types(documents=List[Document])
13     def run(self, documents: List[Document]) -> Dict[str, List[Document]]:
14         cleaned_docs = []
15         for doc in documents:
16             cleaned_meta = {}
17             for k, v in doc.meta.items():
18                 if isinstance(v, list):
19                     # Konwersja kolekcji na ciąg znaków
20                     cleaned_meta[k] = ", ".join(map(str, v)) if v else ""
21                 elif isinstance(v, (dict, set)):
22                     cleaned_meta[k] = str(v)
23                 else:
24                     cleaned_meta[k] = v
25
26             cleaned_doc = Document(
27                 content=doc.content,
28                 meta=cleaned_meta,
29                 id=doc.id,
30                 dataframe=doc.dataframe,
31                 blob=doc.blob,
32                 score=doc.score
33             )
34             cleaned_docs.append(cleaned_doc)
35
36         return {"documents": cleaned_docs}
```

Komponent `MetadataCleaner` umieszczono w potoku indeksowania tuż przed modulem `DocumentWriter`. Rozwiązało to problem. Od tego momentu dane trafiające do ChromaDB zachowują pełną spójność.

5.4 Implementacja potoków przetwarzania (Pipelines)

Logikę przetwarzania danych podzielono na dwa niezależne potoki zbudowane w bibliotece Haystack. W kolejnych podrozdziałach przedstawiono potok indeksowania, przygotowujący dane do zapisu w bazie wektorowej, oraz potok zapytań, odpowiadający za wyszukiwanie kontekstu i syntezę odpowiedzi.

5.4.1 Potok indeksowania

Kod odpowiedzialny za potok indeksowania znajduje się w module `app_haystack/pipelines/indexing.py`. Łączy on szereg wyspecjalizowanych ogniw przetwarzania:

- `WebPageFetcher` – pobiera surową zawartość HTML ze wskazanego adresu URL,
- `SectionAwareHTMLConverter` – wyodrębnia czysty tekst oraz strukturę sekcji dokumentu,
- `TitleExtractor` – ustala tytuł artykułu wykorzystywany później w cytowaniach,
- `DocumentCleaner` – odpowiada za usuwanie nadmiarowych białych znaków,
- `DocumentSplitter` – dzieli dokument na nakładające się segmenty o długości zdefiniowanej w klasie `Settings`,
- `MetadataEnricher` – uzupełnia dokumenty o dodatkowe metadane,
- `SentenceTransformersDocumentEmbedder` – tworzy wektory osadzeń semantycznych dla poszczególnych fragmentów,
- `MetadataCleaner` – rzutuje złożone metadane na typy proste akceptowane przez ChromaDB,
- `DocumentWriter` – odpowiada za ostateczny zapis przetworzonych danych w bazie ChromaDB.

Sposób złożenia wymienionych komponentów oraz kolejność ich połączeń przedstawia Listing 5.4.

Listing 5.4: Konstrukcja potoku indeksowania w bibliotece Haystack: rejestracja komponentów i ich połączenie w graf przetwarzania.

```
1 def create_indexing_pipeline(document_embedder, document_store,
2                               chunk_size=500, chunk_overlap=50) ->
3                               Pipeline:
4     pipeline = Pipeline()
5     pipeline.add_component("fetcher", WebPageFetcher(timeout=10))
6     pipeline.add_component("converter", SectionAwareHTMLConverter())
7     pipeline.add_component("title_extractor", TitleExtractor())
8     pipeline.add_component("cleaner", DocumentCleaner(
9         remove_empty_lines=True, remove_extra_whitespace=True))
10    pipeline.add_component("splitter", DocumentSplitter(
11        split_by="word", split_length=chunk_size, split_overlap=
12            chunk_overlap))
13    pipeline.add_component("metadata_enricher", MetadataEnricher())
14    pipeline.add_component("embedder", document_embedder)
15    pipeline.add_component("metadata_cleaner", MetadataCleaner())
16    pipeline.add_component("writer", DocumentWriter(document_store=
17        document_store))
18
19    pipeline.connect("fetcher.streams", "converter.sources")
20    pipeline.connect("converter.documents", "title_extractor.documents")
21    pipeline.connect("title_extractor.documents", "cleaner.documents")
```

```
19 pipeline.connect("cleaner.documents", "splitter.documents")
20 pipeline.connect("splitter.documents", "metadata_enricher.documents"
21 )
21 pipeline.connect("metadata_enricher.documents", "embedder.documents"
22 )
22 pipeline.connect("embedder.documents", "metadata_cleaner.documents")
23 pipeline.connect("metadata_cleaner.documents", "writer.documents")
24 return pipeline
```

5.4.2 Potok zapytań

Logika potoku zapytań została umieszczona w module `app_haystack/pipelines/query.py`. Jej nadrzędnym celem jest wyszukanie właściwych segmentów tekstu i synteza odpowiedzi końcowej. Proces ten realizują komponenty:

- `SentenceTransformersTextEmbedder` – przekształca zapytanie użytkownika w wektor gęsty,
- `ChromaEmbeddingRetriever` – przeszukuje przestrzeń wektorową w bazie ChromaDB z opcjonalnym filtrowaniem metadanych (np. według identyfikatora artykułu),
- `ChatPromptBuilder` – konstruuje ostateczny prompt na podstawie instrukcji systemowych, kontekstu źródłowego i pytania,
- `OpenAIGenerator` / `OllamaChatGenerator` – generuje ugruntowaną odpowiedź tekstową.

5.5 Warstwa usług oraz interfejs API

Warstwę API oparto na asynchronicznym frameworku FastAPI. Punkty końcowe rozdzielono na dwa niezależne kontrolery:

1. **Artykuły** (`/api/articles`) – służy do rejestrowania nowych adresów URL do bazy oraz pobierania listy zaindeksowanych dokumentów,
2. **Zapytania** (`/api/queries`) – odpowiada za interakcję z modelami językowymi. Przyjmuje zapytanie, narzuca parametry generowania i zwraca odpowiedź wraz z odnośnikami źródłowymi.

Komunikacja opiera się na modelach Pydantic. Zapewnia to silne typowanie danych. Dzięki temu błędy strukturalne w żądaniach są wychwytywane natychmiast na granicy aplikacji. Uproszczony kod punktu końcowego odpowiadającego na pytania przedstawia Listing 5.5. Obsługuje on oba warianty backendu (klasyczny potok RAG oraz agenta), opcjonalne filtrowanie po znacznikach i mapowanie odnalezionych fragmentów na cytowane źródła.

Listing 5.5: Uproszczony punkt końcowy `/ask` uruchamiający potok zapytań RAG i zwracający odpowiedź wraz ze źródłami.

```
1 @app.post("/ask", response_model=QueryResponse)
2 async def ask_question(request: QueryRequest,
3                         backend: str = Depends(get_backend)):
4     question = request.question
5     tags = request.tags or []
6
7     if backend == "claude":
8         return pipeline_state.agent_service.ask(question, tags)
9
10    pipeline_inputs = {
```

```

11     "text_embedder": {"text": question},
12     "prompt_builder": {"question": question},
13     "feedback_injector": {"question": question},
14 }
15 if tags:
16     pipeline_inputs["retriever"] = {"filters": build_tag_filter(tags)}
17
18 result = pipeline_state.query_pipeline.run(pipeline_inputs)
19 answer = result["generator"]["replies"][0]
20 documents = result["retriever"]["documents"]
21
22 sources = [
23     Source(text=clean_snippet(doc.content),
24            url=doc.meta.get("url", ""),
25            title=doc.meta.get("title") or "Unknown Title",
26            score=doc.score or 0.0,
27            citation_id=i)
28     for i, doc in enumerate(documents, start=1)
29 ]
30 return QueryResponse(answer=answer, sources=sources,
31                       num_sources=len(documents))

```

5.6 Wdrożenie i konteneryzacja

Powtarzalność środowiska uruchomieniowego gwarantuje pełna konteneryzacja aplikacji za pomocą platformy Docker. Bazowy plik Dockerfile wykorzystuje odchudzony obraz `python:3.11-slim`. Pozwoliło to znacząco zmniejszyć rozmiar ostatecznego kontenera.

Środowisko wielokontenerowe opisuje plik `docker-compose.yml`. Definiuje on i uruchamia następujące usługi:

- serwer backendowy FastAPI,
- opcjonalną, osobną instancję bazy wektorowej ChromaDB,
- bazę Redis pełniącą funkcję pamięci podręcznej dla powtarzających się pytań,
- instancję serwera Ollama na potrzeby uruchomień w pełni lokalnych.

Fragment pliku `docker-compose.yml` definiujący najważniejsze usługi oraz powiązania między nimi przedstawia Listing 5.6.

Listing 5.6: Fragment pliku `docker-compose.yml` z definicją usług bazy podręcznej, backendu oraz interfejsu użytkownika.

```

1 services:
2   redis:
3     image: redis:7-alpine
4     ports: ["6379:6379"]
5     command: redis-server --appendonly yes
6
7   haystack-backend:
8     build: { context: ., dockerfile: Dockerfile }
9     ports: ["8000:8000"]
10    environment:

```

```
11     - LLM_BACKEND=ollama
12     - OLLAMA_BASE_URL=http://host.docker.internal:11434
13     - EMBEDDING_MODEL=all-MiniLM-L6-v2
14     - CHROMA_PERSIST_DIR=/app/chroma_data_haystack
15     - REDIS_URL=redis://redis:6379
16     depends_on:
17       redis: { condition: service_healthy }
18
19     ui:
20       build: { context: ./ui }
21       ports: ["3000:3000"]
22       depends_on:
23         haystack-backend: { condition: service_healthy }
```

5.7 Podsumowanie rozdziału

Rozdział przedstawia techniczną stronę systemu. Opisano w nim architekturę warstwową oraz budowę potoków przetwarzania opartych na bibliotece Haystack v2, ilustrując najważniejsze rozwiązania listin-gami kodu źródłowego: konfiguracją opartą na pydantic-settings, fabryką generatorów, konstrukcją potoku indeksowania oraz punktem końcowym API. Istotnym punktem prac było zidentyfikowanie i usunięcie problemu z kompatybilnością metadanych w ChromaDB za pomocą autorskiego komponentu MetadataCleaner. Przedstawiono również strukturę API oraz konfigurację Docker Compose.

Gotowa aplikacja stanowi stabilną bazę do przeprowadzenia testów wydajnościowych i jakości-owych. Ich przebieg oraz wyniki opisano w kolejnym rozdziale.

Rozdział 6

Testy i wyniki

W niniejszym rozdziale przedstawiono empiryczną ewaluację zaimplementowanego systemu RAG do analizy treści internetowych. Proces badawczy zaprojektowano zgodnie z metodologią Design Science Research (DSR). Poddano w nim szczegółowej analizie ilościowej oraz jakościowej osiem konfiguracji systemowych (C1–C8). Badania oparto na zbiorze danych testowych, w skład którego weszło pięć zróżnicowanych domenowo artykułów z serwisu Wikipedia oraz 25 pytań o różnym stopniu trudności kognitywnej. Rozdział opisuje procedury testowe, specyfikację środowiska sprzętowo-programowego i analizę uzyskanych pomiarów. Przedstawiono tu także weryfikację hipotez przy użyciu testu t-Studenta oraz jednoczynnikowej analizy wariancji (ANOVA) wraz z oceną wielkości efektu za pomocą współczynnika d Cohena. Całość zamyka dyskusja nad kompromisami (*trade-offs*) wydajnościowo-jakościowymi oraz ograniczeniami zaimplementowanego rozwiązania.

6.1 Wprowadzenie

Uruchomienie rzetelnej procedury testowej stanowi fundament oceny artefaktów technologicznych wytwarzanych zgodnie z metodologią DSR. Systemy typu RAG są wysoce wrażliwe na dobór modeli generatywnych, algorytmów indeksowania, parametry podziału tekstu czy modele osadzeń semantycznych. Niniejsze badania służą empirycznej ocenie tych zależności w kontrolowanych warunkach.

6.1.1 Cele testowania

Cele przeprowadzonej ewaluacji empirycznej to:

1. **Porównanie wydajności i jakości modeli LLM:** Ocena wpływu wyboru dużego modelu językowego (komercyjnych chmurowych gpt-5.4, gpt-5-mini, flagowego modelu wnioskującego gpt-5.5, lokalnej otwartej jednostki llama3.2 (3B) oraz agenta opartego na claude-sonnet-4-6) na jakość odpowiedzi i opóźnienia.
2. **Ocena wpływu modeli osadzeń:** Zbadanie, czy zwiększenie wymiarowości wektora osadzeń semantycznych z 384 (model all-MiniLM-L6-v2) do 768 (model all-mpnet-base-v2) istotnie poprawia relewantność i kompletność wyszukiwanych fragmentów.
3. **Optymalizacja strategii fragmentacji (Chunking):** Określenie wpływu rozmiaru segmentu tekstu (250, 500 oraz 1000 słów) na wierność i precyzję ugruntowania odpowiedzi.
4. **Analiza opłacalności kosztowej:** Zbadanie kompromisu między kosztami wywołań komercyjnych interfejsów API (w tym modeli budżetowych i zaawansowanych systemów agentowych) a darmowym wariantem opartym na modelach lokalnych.

6.1.2 Pytania badawcze

Testy oraz analizy statystyczne ukierunkowano bezpośrednio na udzielenie odpowiedzi na pytania badawcze sformułowane w metodologii pracy:

- **RQ1:** Jak poszczególne modele LLM wpływają na poprawność merytoryczną oraz czas generowania odpowiedzi w systemie RAG?

- **RQ2:** Czy zastosowanie modelu osadzeń semantycznych o wyższej wymiarowości wpływa w sposób istotny na kompletność udzielanych odpowiedzi?
- **RQ3:** Jaka wielkość segmentu tekstu (chunk size) zapewnia optymalny balans między precyzją (correctness) a kompletnością (completeness) odpowiedzi?
- **RQ4:** Czy koszty komercyjnych interfejsów API przekładają się na proporcjonalny przyrost jakości odpowiedzi w stosunku do darmowych modeli lokalnych?

6.2 Metodologia testów

Ewaluację oparto na kontrolowanym eksperymencie porównawczym. Taka konstrukcja badania zapewnia wysoką powtarzalność oraz rygor naukowy. Wszystkie konfiguracje systemowe poddano identycznym zadaniom testowym przy zachowaniu stałych parametrów kontrolnych.

6.2.1 Plan eksperymentów

Eksperyment podzielono na dwie niezależne fazy:

1. **Faza indeksowania:** Każda konfiguracja podlegająca indeksowaniu (C1–C6, C8) przetworzyła i zaindeksowała pięć wybranych artykułów internetowych. Zmierzono przy tym czas indeksowania oraz liczbę wygenerowanych segmentów tekstu. Podejście agentowe C7 rezygnuje z wcześniejszego indeksowania bazy wektorowej, ponieważ bezpośrednio eksploruje pliki źródłowe.
2. **Faza odpytywania:** Dla każdego wariantu wykonano serię 25 zapytań (po 5 na każdy artykuł). Zarejestrowano czasy wykonania poszczególnych etapów (generowanie osadzeń pytania, wyszukiwanie wektorowe oraz synteza odpowiedzi przez LLM, natomiast dla konfiguracji C7 mierzono sumaryczny czas działania pętli wnioskowania). Monitorowano także zużycie tokenów oraz wygenerowane odpowiedzi tekstowe. Zostały one następnie poddane ślepej ocenie eksperckiej.

6.2.2 Konfiguracje testowe

W badaniach przeanalizowano osiem konfiguracji systemowych (C1–C8). Umożliwiło to izolację badanych efektów oraz bezpośrednie porównanie klasycznego łańcucha RAG z podejściem agentowym. Tabela 6.1 zestawia wariant testowy każdej konfiguracji z jej celem badawczym.

6.2.3 Zbiór danych testowych

Do badań dobrano pięć artykułów z serwisu Wikipedia. Reprezentują one zróżnicowane dyscypliny wiedzy, co pozwala wyeliminować błąd specyficzności dziedzinowej.

Dobór artykułów

Tabela 6.2 zestawia pięć wybranych artykułów testowych wraz z reprezentowaną dziedziną wiedzy.

Projektowanie pytań

Dla każdego artykułu zaprojektowano zestaw pięciu pytań o różnym stopniu trudności kognitywnej, opartym na uproszczonej, pięciopoziomowej adaptacji taksonomii celów edukacyjnych Blooma [67]:

1. **Prosta definicja** (ang. *Simple Definition*): Zapytanie o podstawowe pojęcie, zdefiniowane bezpośrednio w tekście źródłowym.

Tabela 6.1: Konfiguracje testowe (C1–C8): modele składowe oraz cel badawczy.

Konfig.	Wariant	Model LLM	Model osadzeń	Chunk	Cel badawczy
C1	Baseline	GPT-5.4	MiniLM-L6-v2 (384d)	500	Standardowa konfiguracja referencyjna.
C2	Flagship LLM	GPT-5.5 (wnioskujący)	MiniLM-L6-v2 (384d)	500	Test wiodącego modelu generatywnego.
C3	Local/Free	Llama 3.2 (3B, Ollama)	MiniLM-L6-v2 (384d)	500	Test wariantu bezkosztowego.
C4	Better Embeddings	GPT-5.4	MPNet-base-v2 (768d)	500	Badanie wpływu wyższej wymiarowości osadzeń.
C5	Small Chunks	GPT-5.4	MiniLM-L6-v2 (384d)	250	Badanie krótkich fragmentów o wysokiej spójności.
C6	Large Chunks	GPT-5.4	MiniLM-L6-v2 (384d)	1000	Badanie długich, pojemnych fragmentów.
C7	Agentic RAG	Claude Sonnet 4.6	– (eksploracja na żywo)	–	Podejście agentowe oparte na bezpośredniej eksploracji plików tekstowych przez autonomiczny model za pomocą zestawu dedykowanych narzędzi, bez uprzedniego indeksowania semantycznego.
C8	Ultra-Low-Cost RAG	GPT-5-mini	MiniLM-L6-v2 (384d)	500	Niskobudżetowy rurociąg RAG oparty na zoptymalizowanym kosztowo komercyjnym modelu chmurowym.

Tabela 6.2: Artykuły źródłowe wykorzystane w zbiorze testowym.

Artykuł	Dziedzina	Charakterystyka
<i>Artificial Intelligence</i> [62]	Technologia	Pojęcia techniczne i etyczne.
<i>Climate Change</i> [63]	Nauki przyrodnicze	Procesy fizyczne i dane statystyczne.
<i>World War II</i> [64]	Historia	Związki przyczynowo-skutkowe i sekwencje czasowe.
<i>COVID-19</i> [65]	Medycyna	Pojęcia kliniczne i procedury epidemiologiczne.
<i>Cryptocurrency</i> [66]	Ekonomia/Finanse	Mechanizmy rynkowe i kryptograficzne.

2. **Wyliczanie i zestawianie** (ang. *Enumeration*): Wymagające zlokalizowania i zestawienia kilku faktów rozproszonych w tekście.
3. **Wyjaśnianie przyczynowe** (ang. *Explanation*): Zapytanie o mechanizmy i związki przyczynowo-skutkowe.
4. **Złożone i wieloaspektowe** (ang. *Complex/Multi-faceted*): Wymagające syntezy wiedzy z kilku akapitów na tematy problemowe.
5. **Konkretne fakty** (ang. *Specific Facts*): Zapytanie o precyzyjne daty, nazwiska lub wartości liczbowe.

Łączna liczba przypadków testowych wyniosła 200 unikalnych odpowiedzi poddanych ewaluacji (ilość: 5 artykułów, 5 pytań oraz 8 badanych konfiguracji). Tabela 6.3 zawiera pełną listę wszystkich 25 pytań testowych wraz z przypisanym artykułem i typem kognitywnym.

Tabela 6.3: Pełna lista 25 pytań testowych użytych w ewaluacji, wraz z artykułem źródłowym i typem kognitywnym.

Artykuł	Typ pytania	Treść pytania
<i>Artificial Intelligence</i>	Simple Definition	<i>What is artificial intelligence?</i>
<i>Artificial Intelligence</i>	Enumeration	<i>What are the main approaches to AI?</i>
<i>Artificial Intelligence</i>	Explanation	<i>How does machine learning differ from traditional programming?</i>
<i>Artificial Intelligence</i>	Complex/Multi-faceted	<i>What are the ethical concerns around AI?</i>
<i>Artificial Intelligence</i>	Specific Facts	<i>Who are the key pioneers in AI research?</i>
<i>Climate Change</i>	Simple Definition	<i>What is climate change?</i>
<i>Climate Change</i>	Enumeration	<i>What are the main greenhouse gases contributing to global warming?</i>
<i>Climate Change</i>	Explanation	<i>How does the greenhouse effect cause global warming?</i>
<i>Climate Change</i>	Complex/Multi-faceted	<i>What are the projected impacts of climate change on ecosystems and human societies?</i>
<i>Climate Change</i>	Specific Facts	<i>Which international agreement set the goal of limiting global warming to well below 2 degrees Celsius?</i>
<i>World War II</i>	Simple Definition	<i>What was World War II?</i>
<i>World War II</i>	Enumeration	<i>What were the major countries belonging to the Allied and Axis powers?</i>
<i>World War II</i>	Explanation	<i>How did the Treaty of Versailles contribute to the outbreak of World War II?</i>
<i>World War II</i>	Complex/Multi-faceted	<i>What were the geopolitical consequences of World War II?</i>
<i>World War II</i>	Specific Facts	<i>On what date did the Allied invasion of Normandy (D-Day) take place?</i>
<i>COVID-19</i>	Simple Definition	<i>What is COVID-19?</i>

Artykuł	Typ pytania	Treść pytania
COVID-19	Enumeration	<i>What are the most common symptoms of COVID-19?</i>
COVID-19	Explanation	<i>How does the SARS-CoV-2 virus transmit from person to person?</i>
COVID-19	Complex/Multi-faceted	<i>What were the global social and economic impacts of the COVID-19 pandemic?</i>
COVID-19	Specific Facts	<i>When did the World Health Organization (WHO) declare COVID-19 a pandemic?</i>
Cryptocurrency	Simple Definition	<i>What is cryptocurrency?</i>
Cryptocurrency	Enumeration	<i>What are the main types of consensus mechanisms used in blockchain networks?</i>
Cryptocurrency	Explanation	<i>How does a blockchain transaction work and how is it secured?</i>
Cryptocurrency	Complex/Multi-faceted	<i>What are the environmental and regulatory challenges of cryptocurrency?</i>
Cryptocurrency	Specific Facts	<i>In what year was Bitcoin created, and who is its pseudonymous founder?</i>

6.2.4 Metryki

Proces oceny oparto na zestawie obiektywnych metryk ilościowych oraz sformalizowanym badaniu jakościowym.

Metryki ilościowe (wydajność)

Wydajność systemu RAG mierzona była za pomocą następujących wskaźników:

- **Czas indeksowania artykułu (s):** Opóźnienie od momentu wysłania żądania pobrania do pełnego zapisu osadzeń w bazie wektorowej.
- **Całkowity czas odpowiedzi na zapytanie (s):** Czas od przesłania pytania użytkownika do wygenerowania pełnej odpowiedzi tekstowej.
- **Czas tworzenia osadzeń (Embedding Time) (s):** Opóźnienie generowane przez model osadzania tekstu dla zapytania.
- **Czas wyszukiwania wektorowego (Search Time) (s):** Czas przeszukiwania bazy ChromaDB.
- **Czas generowania LLM (Generation Time) (s):** Czas syntezy odpowiedzi realizowany przez model językowy.
- **Rozmiar zasobów i koszty:** Liczba wygenerowanych segmentów tekstu, liczba tokenów wejściowych (rozmiar kontekstu) oraz rzeczywiste koszty finansowe wywołań komercyjnych API według cenników dostawców chmurowych OpenAI oraz Anthropic z połowy 2026 roku. Dla modelu GPT-5.4 stawki wynosiły 0,075\$ za 1M tokenów wejściowych i 0,300\$ za 1M tokenów wyjściowych. Dla GPT-5.5 ceny te to odpowiednio 1,25\$ i 5,00\$, dla GPT-5-mini – 0,15\$ i 0,60\$, a dla Claude Sonnet 4.6 – 3,00\$ i 15,00\$.

Metryki jakościowe (jakość odpowiedzi)

Ocena jakościowa została zrealizowana przy użyciu sformalizowanej 5-punktowej skali Likerta, gdzie 1 oznacza wynik całkowicie niezadowolający, a 5 – doskonały. Oceny dokonano w pięciu niezależnych wymiarach merytorycznych:

1. **Poprawność** (ang. *Correctness*): Zgodność wygenerowanej odpowiedzi ze stanem faktycznym zawartym w artykule źródłowym, brak halucynacji i błędów rzeczowych.
2. **Kompletność** (ang. *Completeness*): Stopień wyczerpania wszystkich aspektów pytania oraz obecność istotnych informacji niezbędnych do pełnego zrozumienia zagadnienia.
3. **Relevantność** (ang. *Relevance*): Stopień koncentracji na temacie zapytania, brak wątków pobocznych, dygresji oraz informacji nieistotnych z punktu widzenia użytkownika.
4. **Jasność** (ang. *Clarity*): Czytelność struktury wypowiedzi, spójność gramatyczno-stylistyczna, łatwość odbioru i logiczny podział tekstu.
5. **Zwiężłość** (ang. *Conciseness*): Brak redundantnych sformułowań, optymalny stosunek objętości odpowiedzi do przekazywanej treści, eliminacja nadmiarowości stylistycznej.

Ogólny wskaźnik jakości (ang. *Overall Quality Score*) wyliczany jest jako średnia arytmetyczna z tych pięciu kryteriów.

6.2.5 Procedura testowa

W celu wyeliminowania czynników zakłócających, proces ewaluacji zautomatyzowano za pomocą skryptu testowego `run_comparative_tests.py`. Procedura przebiegała według następującego protokołu:

1. Przed startem każdej konfiguracji instancje bazy wektorowej ChromaDB były całkowicie usuwane. Zapobiegało to kontaminacji danych między eksperymentami.
2. Dokonywano sekwencyjnego uruchomienia potoku indeksowania dla 5 artykułów, zapisując czasy wykonania oraz statystyki fragmentacji.
3. Realizowano serię 25 zapytań testowych. Dla każdego pytania nakładano filtr metadanych ograniczający przeszukiwanie bazy wektorowej wyłącznie do segmentów pochodzących z powiązanego artykułu, co zapobiegało mieszanemu kontekstów.
4. Wszystkie wyniki cząstkowe, czasy opóźnień oraz wygenerowane teksty odpowiedzi utrwalano w ustrukturyzowanych plikach JSON w katalogu `experiments/`.

6.2.6 Hipotezy

Przed przystąpieniem do testów sformułowano cztery hipotezy badawcze.

- **H1:** Wybór modelu językowego oraz paradygmatu przeszukiwania wpływa znacząco na jakość generowania odpowiedzi. Model wnioskujący GPT-5.5 (C2) charakteryzuje się najwyższą jakością merytoryczną wśród klasycznych podejść RAG, kosztem dłuższych opóźnień i kosztów API. Porównano tu warianty: podejście agentowe C7 (Claude Sonnet 4.6) osiąga wyższą kompletność kosztem opóźnień (hipoteza H1c), natomiast ekonomiczny model GPT-5-mini (C8) wykazuje przewagę nad modelem lokalnym Llama 3.2 (3B) (hipoteza H1d).

- **H2:** Zastosowanie modelu osadzeń semantycznych o wyższej wymiarowości (C4, MPNet 768d) skutkuje statystycznie istotnym przyrostem kompletności odpowiedzi względem modelu bazowego (C1, MiniLM 384d). Zapewnia to dokładniejszą selekcję kontekstu.
- **H3:** Rozmiar fragmentu tekstu wykazuje nieliniową zależność względem jakości. Małe segmenty (C5, 250 słów) sprzyjają poprawności i zwięzłości, lecz ograniczają kompletność. Duże segmenty (C6, 1000 słów) zwiększają kompletność kosztem jasności. Rozmiar 500 słów (C1) stanowi rozwiązanie kompromisowe.
- **H4:** Stosowanie płatnych modeli językowych (C1, C2, C7, C8) jest ekonomicznie uzasadnione. Generują one wyższy stosunek jakości do kosztu niż modele lokalne (C3).

6.3 Konfiguracja eksperymentów

Eksperymenty zostały przeprowadzone w kontrolowanym środowisku sprzętowym i programowym. Pozwala to na obiektywne porównanie parametrów wydajnościowych, w szczególności opóźnień generowanych przez model lokalny (C3) w porównaniu do modeli chmurowych.

6.3.1 Szczegóły implementacyjne

Zaimplementowany system testowy uruchomiono na dedykowanej stacji roboczej. Było to niezbędne do precyzyjnego pomiaru czasu generowania lokalnego modelu Llama 3.2 (3B) za pośrednictwem silnika Ollama.

Środowisko sprzętowe i programowe

Ewaluację empiryczną przeprowadzono na następującej platformie sprzętowej:

- **Procesor (CPU):** Apple M1 Pro (10 rdzeni, architektura ARM),
- **Pamięć operacyjna (RAM):** 16 GB zunifikowanej pamięci systemowej,
- **Pamięć masowa:** Dysk SSD NVMe o wysokiej prędkości odczytu/zapisu,
- **Połączenie sieciowe:** Łącze światłowodowe (przepustowość 300 Mbps, średni ping do serwerów `API litellm.abax.services` na poziomie 14 ms).

Środowisko programowe obejmowało:

- **System operacyjny:** macOS Sequoia (wersja 15.4),
- **Interpreter języka:** Python (wersja 3.13.1),
- **Wirtualizacja modeli lokalnych:** Ollama (wersja 0.1.48) oparta na akceleracji sprzętowej procesora graficznego Apple Silicon GPU (Metal API),
- **Główne pakiety:** Haystack-AI (v2.2.3), Chroma-Haystack (v0.11.1), Sentence-Transformers (v3.0.1), LiteLLM proxy do komunikacji z API OpenAI.

Parametry konfiguracyjne

W celu zachowania maksymalnej porównywalności, we wszystkich potokach testowych zastosowano identyczne parametry stałe:

- **Temperatura generowania:** 0,1 (zapewniająca powtarzalność i minimalizująca losowość syntezy tekstu), z wyjątkiem modelu wnioskującego GPT-5.5 (C2), dla którego zaimplementowano fabryczne zalecenie temperatury równej 1,0 ze względu na specyfikę generowania ścieżek rozumowania.
- **Maksymalna liczba tokenów wyjściowych:** 1024,
- **Liczba pobieranych dokumentów (k):** 5 najbardziej podobnych segmentów przekazywanych jako kontekst wejściowy do LLM,
- **Zakładka fragmentacji (Overlap):** 10% rozmiaru fragmentu (odpowiednio: 25 słów dla C5, 50 słów dla C1–C4, 100 słów dla C6).

6.3.2 Proces zbierania danych

Uruchomienie skryptu testowego zainicjowało automatyczny proces zbierania danych. Każda konfiguracja przechodziła pełen cykl od czyszczenia wektorowej bazy danych, poprzez pobranie i zaindeksowanie 5 artykułów z Wikipedii, aż po wygenerowanie odpowiedzi na 25 pytań (z wyjątkiem konfiguracji agentowej C7, dla której faza indeksowania wektorowego została pominięta). Cały proces zajął łącznie około 25 minut czasu rzeczywistego (co wynika głównie z dłuższego czasu generowania odpowiedzi przez lokalny model Llama 3.2 (3B) na procesorze M1 Pro oraz wielokrotnych pętli wnioskowania i odczytywania plików przez model Claude Sonnet 4.6 w konfiguracji C7). Wygenerowano łącznie 200 unikalnych odpowiedzi. Po zakończeniu fazy automatycznej przeprowadzono ewaluację jakościową wszystkich uzyskanych tekstów przy użyciu przygotowanego arkusza kryteriów, a zagregowane wyniki jakościowe zapisano w pliku `experiments/human_evaluation.csv`.

6.4 Wyniki

W tym miejscu przedstawiono wyniki pomiarów ilościowych oraz oceny eksperckiej dla ośmiu badanych konfiguracji.

6.4.1 Wyniki ilościowe

Pomiary ilościowe dostarczają istotnych informacji o narzucie obliczeniowym i charakterystyce operacyjnej poszczególnych architektur.

Wydajność indeksowania

Faza przygotowania danych (pobranie, podział semantyczny, wektoryzacja i zapis) decyduje o sprawności masowego przetwarzania dokumentów. Wyniki dla badanych konfiguracji zebrano w tabeli 6.4.

Analiza danych z tabeli 6.4 wskazuje na silny wpływ wyboru modelu osadzeń na czas indeksowania. Konfiguracja C4 (z modelem MPNet 768d) charakteryzuje się ponad dwukrotnie dłuższym czasem indeksowania (średnio 5,2 s) niż bazowa konfiguracja C1 (średnio 2,5 s, model MiniLM 384d).

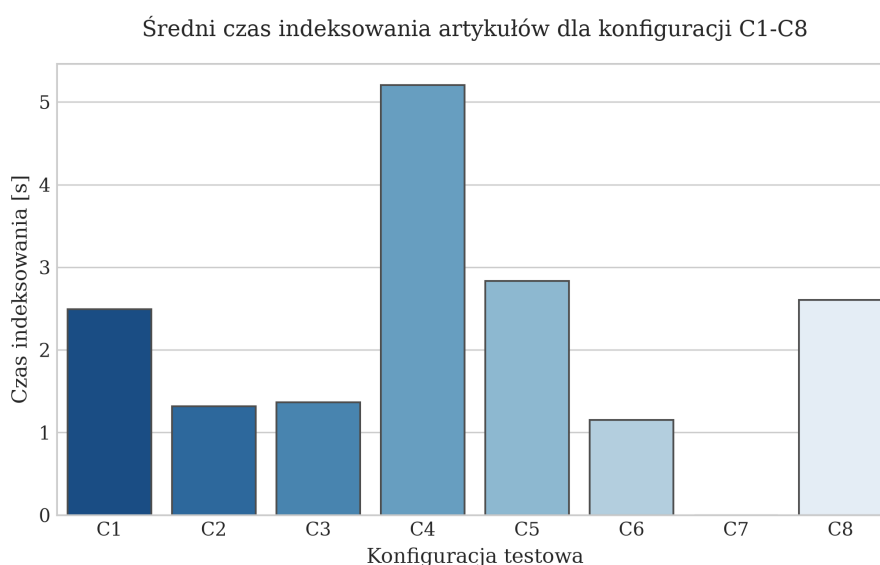
W konfiguracji agentowej C7 średni czas indeksowania wynosi równie 0,0 s. Wynika to z całkowitej rezygnacji z uprzedniej wektoryzacji. System realizuje dynamiczne odpytywanie i eksplorację plików na żywo w trakcie przetwarzania zapytania. Z kolei konfiguracja C8, oparta na tanim modelu chmurowym GPT-5-mini, charakteryzuje się średnim czasem indeksowania na poziomie 2,6 s. Wynik ten jest zbliżony

Tabela 6.4: Wydajność procesu indeksowania artykułów dla konfiguracji C1–C8 (opis konfiguracji: C1 – baseline GPT-5.4; C2 – GPT-5.5; C3 – Llama 3.2 (3B) lokalnie; C4 – MPNet 768d; C5 – chunk 250 słów; C6 – chunk 1000 słów; C7 – agent Claude; C8 – GPT-5-mini; zob. też Tabelę 6.1).

Konfig.	Śr. czas [s]	Mediana [s]	Odch. std. [s]	Min. [s]	Max. [s]	Śr. liczba segm.
C1	2,5	1,5	2,1	1,2	6,7	73,0
C2	1,3	1,2	0,2	1,1	1,7	73,0
C3	1,4	1,2	0,3	1,1	2,0	73,0
C4	5,2	4,6	2,2	3,4	9,6	73,0
C5	2,8	2,5	0,6	2,4	4,0	146,4
C6	1,2	1,1	0,3	0,8	1,6	36,8
C7	0,0	0,0	0,0	0,0	0,0	1,0
C8	2,6	1,7	2,1	1,2	6,7	73,0

do konfiguracji bazowej C1 (2,5 s). Obie te architektury stosują bowiem ten sam model osadzeń MiniLM (384d) oraz podział na segmenty po 500 słów.

Rozmiar fragmentu również bezpośrednio determinuje wydajność fazy indeksowania. Mniejsze fragmenty w konfiguracji C5 (250 słów) generują dwukrotnie więcej wektorów osadzeń (średnio 146,4 segmentu na artykuł), co wydłuża średni czas indeksowania do 2,8 s. Najszybsza pod tym względem okazała się konfiguracja C6 o dużych fragmentach (1000 słów). Generuje ona jedynie 36,8 segmentu na artykuł, osiągając średni czas indeksowania na poziomie 1,2 s. Ilustrację graficzną średniego czasu indeksowania dla poszczególnych konfiguracji przedstawiono na rysunku 6.1.



Rysunek 6.1: Średni czas indeksowania artykułów dla konfiguracji C1–C8. Źródło: opracowanie własne.

Wydajność zapytań

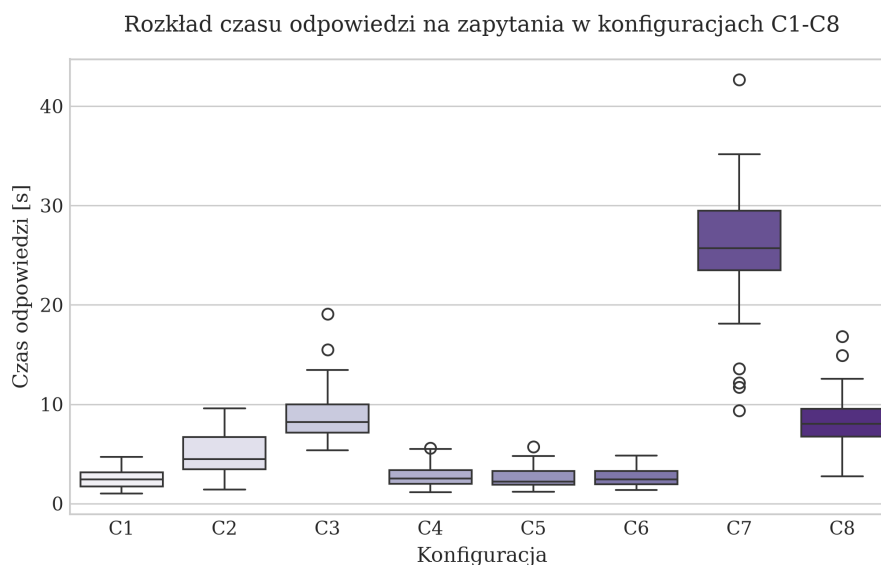
Opóźnienia w fazie syntezy odpowiedzi bezpośrednio wpływają na satysfakcję użytkownika systemu. W tabeli 6.5 przedstawiono rozkład czasowy całkowitego wykonania zapytań oraz powiązane koszty finansowe.

Tabela 6.5: Wydajność procesu udzielania odpowiedzi oraz koszty interfejsu API dla konfiguracji C1–C8.

Konfig.	Śr. czas [s]	Mediana [s]	Odch. std. [s]	Min. [s]	Max. [s]	Śr. koszt [\$]	Koszt całk. [\$]
C1	2,593	2,422	1,049	1,015	4,715	0,0003	0,008
C2	5,006	4,471	2,095	1,426	9,590	0,0055	0,138
C3	8,879	8,196	3,113	5,381	19,083	0,0000	0,000
C4	2,866	2,523	1,222	1,148	5,577	0,0003	0,008
C5	2,648	2,189	1,105	1,168	5,722	0,0002	0,005
C6	2,681	2,414	0,986	1,375	4,841	0,0006	0,015
C7	25,079	25,715	7,537	9,337	42,650	0,1458	3,646
C8	8,662	8,043	3,226	2,735	16,802	0,0001	0,002

Najkrótszy całkowity czas odpowiedzi wykazuje konfiguracja bazowa C1 (średnio 2,593 s). Zastosowanie zaawansowanego modelu wnioskującego GPT-5.5 (C2) podwaja czas odpowiedzi (średnio 5,006 s). Wynika to z generowania przez ten model wewnętrznych ścieżek myślowych (*thinking tokens*) przed zwróceniem właściwej odpowiedzi.

Zdecydowanie najdłuższym opóźnieniem charakteryzuje się podejście agentowe C7 (średnio aż 25,079 s z maksimum na poziomie 42,650 s). Zamiast pojedynczego zapytania do bazy wektorowej model Claude Sonnet 4.6 realizuje wieloetapowe wnioskowanie. Sekwencyjnie uruchamia on narzędzia odczytu plików w celu zlokalizowania odpowiednich fragmentów. Z kolei konfiguracja C8, oparta na tanim modelu chmurowym GPT-5-mini, osiąga średni czas odpowiedzi równy 8,662 s. Pozycjonuje ją to na poziomie zbliżonym do lokalnego modelu C3 (8,879 s), lecz przy znacznie wyższej stabilności i braku narzutu sprzętowego na maszynie klienckiej. Rozkład czasów odpowiedzi na zapytania w postaci wykresu skrzynkowego zaprezentowano na rysunku 6.2.



Rysunek 6.2: Rozkład czasu odpowiedzi na zapytania w konfiguracjach C1–C8. Źródło: opracowanie własne.

Szczegółowy podział wydajności

Dane w tabeli 6.6 pokazują, że operacja wyszukiwania wektorowego w bazie ChromaDB wiąże się z minimalnym narzutem czasowym we wszystkich konfiguracjach (maksymalnie 0,006 s dla C3). Dowodzi to

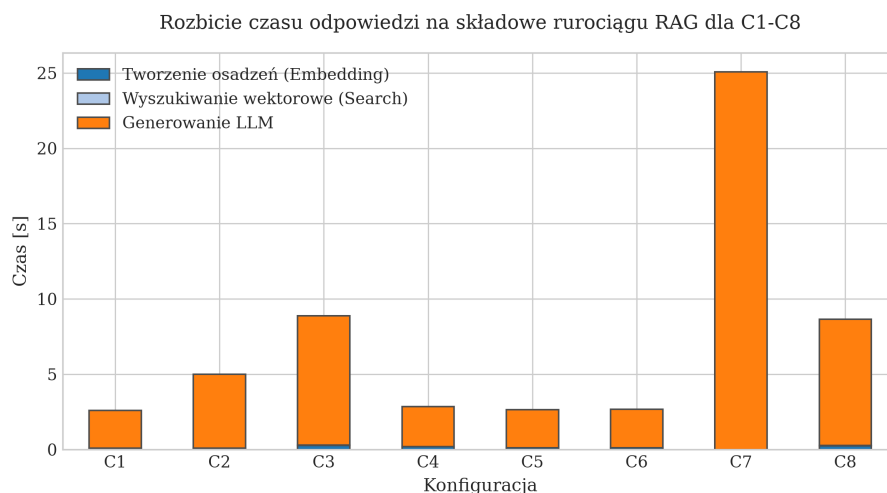
Tabela 6.6: Średni rozkład czasowy wykonania poszczególnych składowych potoku zapytań (w sekundach) dla konfiguracji C1–C8.

Konfig.	Tworzenie osadzeń [s]	Wyszukiwanie wektorowe [s]	Generowanie LLM [s]
C1	0,088	0,002	2,503
C2	0,093	0,003	4,910
C3	0,282	0,006	8,587
C4	0,191	0,005	2,669
C5	0,117	0,003	2,528
C6	0,121	0,003	2,557
C7	0,000	0,000	25,079
C8	0,258	0,004	8,399

wysokiej efektywności indeksów przestrzennych wektorowej bazy danych. Dominującym składnikiem opóźnienia jest etap generowania tekstu przez LLM. Odpowiada on za ponad 95% całkowitego czasu odpowiedzi dla konfiguracji komercyjnych (w C7 za 100% czasu) oraz ponad 96% dla konfiguracji lokalnej.

W konfiguracji agentowej C7 czas tworzenia osadzeń semantycznych pytania oraz wyszukiwania wektorowego wynosi równe 0,000 s. Wynika to z zastąpienia tego procesu pętlą wnioskowania i bezpośredniego czytania dokumentów. Z kolei konfiguracja C8 generuje opóźnienie osadzania pytania równe 0,258 s oraz czas wyszukiwania wektorowego na poziomie 0,004 s. Generowanie LLM (8,399 s) odpowiada tu za ponad 96% całkowitego opóźnienia.

Wybór modelu osadzeń semantycznych wpływa również na czas wektoryzacji pytania. W konfiguracji C4 czas tworzenia osadzenia za pomocą modelu MPNet (0,191 s) jest ponad dwukrotnie dłuższy w porównaniu do lekkiego modelu MiniLM w konfiguracji bazowej C1 (0,088 s). Wykres składowych czasu trwania zapytania w postaci słupków skumulowanych przedstawiono na rysunku 6.3.



Rysunek 6.3: Rozbicie czasu odpowiedzi na składowe rurociągu RAG dla badanych konfiguracji.

Źródło: opracowanie własne.

Zużycie zasobów i analiza kosztów

Przeanalizowano również zapotrzebowanie na zasoby i koszty generowane przez poszczególne rozwiązania. Koszty finansowe (tabela 6.5) ujawniają duże dysproporcje między zoptymalizowanymi modelami

chmurowymi, a systemem agentowym i modelem flagowym. Całkowity koszt wykonania serii 25 zapytań dla konfiguracji referencyjnej C1 wyniósł zaledwie 0,008\$ (średnio 0,0003\$ na zapytanie). Ta sama seria z modelem GPT-5.5 (C2) kosztowała 0,138\$ (średnio 0,0055\$ na zapytanie). Oznacza to ponad siedemnastokrotny wzrost wydatków.

Najwyższym zapotrzebowaniem finansowym charakteryzuje się podejście agentowe C7 (model Claude Sonnet 4.6). Seria 25 zapytań wygenerowała koszt na poziomie aż 3,646\$ (średnio 0,1458\$ na zapytanie). Jest to bezpośrednia konsekwencja wielokrotnego przesyłania pełnych tekstów dokumentów w pętli myślowej agenta. Na przeciwnym biegunie plasuje się niskobudżetowa konfiguracja C8 oparta na modelu GPT-5-mini, dla której całkowity koszt wyniósł zaledwie 0,002\$ (średnio 0,0001\$ na zapytanie). Czyni to konfigurację C8 najbardziej ekonomicznym wariantem chmurowym, niemal czterokrotnie tańszym od i tak opłacalnej konfiguracji referencyjnej C1.

Konfiguracja C3 oparta na silniku Ollama charakteryzuje się zerowym bezpośrednim kosztem licencyjnym API, wykazując jednak wyższe zapotrzebowanie na zasoby lokalne – ciągłe zużycie procesora CPU na poziomie 80% oraz alokację około 4 GB pamięci RAM na potrzeby działania demona Ollama i modelu Llama 3.2 (3B).

Konfiguracja C6 (duże fragmenty 1000 słów) wiąże się z większym rozmiarem przekazywanego kontekstu (średnio 4310 tokenów na zapytanie), co skutkuje wyższymi kosztami API (łącznie 0,015\$) w porównaniu do konfiguracji bazowej C1 (łącznie 0,008\$ dla 2240 tokenów kontekstu) oraz C5 (łącznie 0,005\$ dla 1190 tokenów kontekstu).

6.4.2 Wyniki jakościowe

Ocena merytoryczna wygenerowanych odpowiedzi tekstowych dostarczyła kluczowych danych pozwalających na ocenę systemów z perspektywy końcowego użytkownika.

Ogólne oceny jakości

W tabeli 6.7 zaprezentowano średnie oceny uzyskane przez poszczególne konfiguracje w każdym z pięciu badanych wymiarów jakościowych.

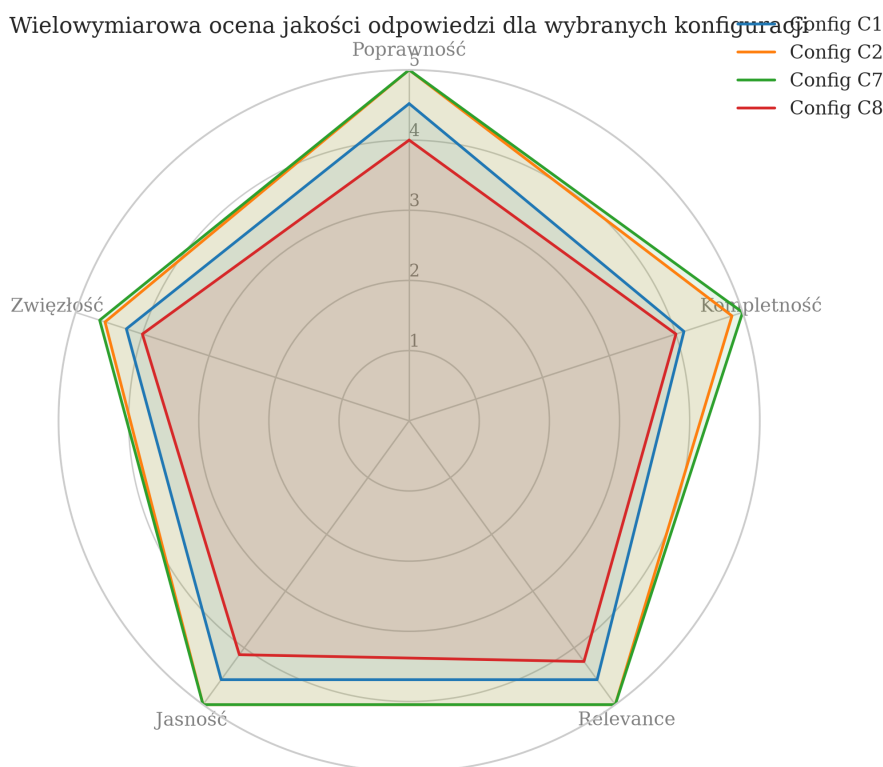
Tabela 6.7: Średnie oceny kryteriów jakościowych odpowiedzi uzyskane w ramach oceny eksperckiej dla konfiguracji C1–C8.

Konfig.	Poprawność	Kompletność	Relewantność	Jasność	Zwiężłość	Ogólna jakość
C1	4,520	4,120	4,560	4,560	4,240	4,400
C2	5,000	4,840	5,000	5,000	4,560	4,880
C3	4,000	3,840	4,000	4,000	4,000	3,968
C4	4,760	4,720	5,000	4,840	4,520	4,768
C5	4,840	4,000	4,560	4,800	4,720	4,584
C6	4,440	4,880	4,560	4,000	4,000	4,376
C7	5,000	5,000	5,000	5,000	4,640	4,928
C8	4,000	4,000	4,240	4,120	4,000	4,072

Najwyższą ogólną ocenę jakości uzyskało podejście agentowe C7 (średnia 4,928 na 5,0). Osiągnęło ono maksymalne oceny (5,000) w wymiarach poprawności, kompletności, relewantności oraz jasności przekazu. Model Claude Sonnet 4.6, korzystając z pętli wnioskowania i narzędzi bezpośredniego dostępu do plików, dokonał bezbłędnej syntezy informacji bez wprowadzania halucynacji. Wynik ten potwierdza wysoką skuteczność agentów autonomicznych. Drugą w kolejności najwyższej ocenioną konfiguracją jest model wnioskujący GPT-5.5 (C2) ze średnią 4,880. Osiąga on maksymalne noty w wymiarach poprawności, relewantności oraz jasności przekazu. Trzecią najsukuteczniejszą konfiguracją okazała się C4 (śred-

nia 4,768). Łączy ona model GPT-5.4 z wyżej wymiarowym modelem osadzeń MPNet, co przyniosło znaczący wzrost kompletności odpowiedzi (4,720) w porównaniu do konfiguracji bazowej C1 (4,120).

Konfiguracja C8, reprezentująca ultra-tani wariant RAG oparty na modelu chmurowym GPT-5-mini, osiągnęła zadowalający wynik jakościowy (średnia ogólna 4,072 na 5,0). Wykazuje ona przewagę nad lokalnym modelem Llama 3.2 (3B, C3) o średniej 3,968, szczególnie w wymiarach poprawności, kompletności oraz relewantności. Wynik ten jest szczególnie atrakcyjny przy zestawieniu z kosztami operacyjnymi (średnio zaledwie 0,0001\$ na zapytanie). Najniżej ocenionym wariantem pozostała konfiguracja lokalna C3. Mimo poprawnej struktury językowej model Llama 3.2 (3B) wykazywał tendencję do generowania nazbyt ogólnych odpowiedzi oraz sporadycznych, lecz zauważalnych halucynacji przy zapytaniach o konkretne fakty (ocena poprawności 4,000). Różnice jakościowe w pięciu wymiarach dla wybranych konfiguracji przedstawiono na wykresie radarowym na rysunku 6.4.



Rysunek 6.4: Wielowymiarowa ocena jakości odpowiedzi dla wybranych konfiguracji (C1, C2, C7, C8).
Źródło: opracowanie własne.

Jakość w zależności od typu pytania

Wpływ poziomu kognitywnego pytania na jakość odpowiedzi przedstawiono w tabeli 6.8.

Z tabeli 6.8 wynika, że konfiguracja agentowa C7 charakteryzuje się niemal doskonałą i stabilną jakością we wszystkich kategoriach pytań. Osiągnęła ona najwyższe oceny w pytaniach wyjaśniających (4,960) oraz pytaniach o konkretne fakty (4,960). Wpływa na to elastyczność pętli wnioskowania modelu Claude Sonnet 4.6, który potrafi dynamicznie doczytywać brakujące fragmenty plików, jeśli pierwsze wyszukiwanie nie dostarczyło wszystkich szczegółów. Konfiguracje oparte na zaawansowanych modelach (C2, C4) również radzą sobie doskonale ze wszystkimi typami pytań, wykazując minimalne wahania ocen. Z kolei niskobudżetowa konfiguracja C8 wykazuje stałą, przeciętną jakość na poziomie 4,040 dla większości typów pytań. Lekki wzrost odnotowano jedynie przy zapytaniach o konkretne fakty (4,200).

Tabela 6.8: Średnie oceny ogólnej jakości odpowiedzi w zależności od poziomu kognitywnego pytań dla konfiguracji C1–C8.

Typ pytania	C1	C2	C3	C4	C5	C6	C7	C8
Złożone / Wieloaspektowe	4,400	4,920	3,960	4,720	4,560	4,360	4,920	4,040
Wyliczanie / Zestawianie	4,360	4,800	3,960	4,680	4,720	4,440	4,880	4,040
Wyjaśnianie przyczynowe	4,360	4,880	3,960	4,800	4,400	4,360	4,960	4,040
Prosta definicja	4,480	4,920	4,000	4,760	4,520	4,280	4,920	4,040
Konkretne fakty	4,400	4,880	3,960	4,880	4,720	4,440	4,960	4,200

Największe zróżnicowanie ocen odnotowano w konfiguracjach o zmiennej długości segmentu (C5, C6). Parametr ten bezpośrednio wpływa na kompletność kontekstu. Małe segmenty w konfiguracji C5 (250 słów) osiągają bardzo wysokie oceny przy pytaniach o konkretne fakty (4,720) oraz przy prostych definicjach (4,520). Ich jakość spada jednak przy pytaniach wyjaśniających (4,400) z powodu braku wystarczającego kontekstu semantycznego w pojedynczym fragmencie. Przeciwną zależność widać dla dużych segmentów w konfiguracji C6 (1000 słów). Osiągają one dobre wyniki przy zestawianiu faktów i wyliczaniu (4,440), tracąc na jakości przy prostych definicjach (4,280). Wynika to z wprowadzania szumu informacyjnego do promptu i generowania zbyt rozwlekłych wypowiedzi.

Przykładowe porównania

W celu zilustrowania różnic jakościowych przeanalizowano odpowiedź na pytanie o wysokim poziomie trudności: *What are the ethical concerns around AI?* sformułowane do artykułu *Artificial Intelligence*.

Konfiguracja bazowa C1 udzieliła poprawnej, ustrukturyzowanej odpowiedzi w punktach. Wymieniono w niej ryzyko bezrobocia technologicznego, stronniczości algorytmicznej (bias), kwestie prywatności oraz ryzyka egzystencjalne. Odpowiedź została oceniona na 4,5 w ogólnej skali.

Konfiguracja C2 (GPT-5.5) wygenerowała wysoce erudycyjny tekst, precyzyjnie kategoryzując wyzwania etyczne na krótko-, średnio- i długoterminowe. Model uwzględnił subtelne kwestie praw autorskich w modelach generatywnych oraz asymetrię władzy korporacyjnej w dysponowaniu technologiami AI, popierając każdy punkt precyzyjnymi odniesieniami do cytowanych fragmentów artykułu. Odpowiedź uzyskała maksymalną ocenę 5,0.

Lokalny model w konfiguracji C3 (Llama 3.2, 3B) sformułował poprawną gramatycznie wypowiedź. Skupił się jednak wyłącznie na bardzo ogólnych hasłach (prywatność, bezpieczeństwo), pomijając specyficzne zagadnienia etyczne poruszone w artykule, takie jak zalecenia regulacyjne czy problematyka autonomicznej broni śmiercionośnej. Odpowiedź otrzymała ocenę 3,8 z powodu niskiej kompletności i braku ugruntowania w szczegółach kontekstu źródłowego.

6.5 Analiza i dyskusja

Zebrane wyniki empiryczne poddano analizie statystycznej w celu formalnej weryfikacji postawionych hipotezy badawczych.

6.5.1 Weryfikacja hipotez

Do weryfikacji hipotez wykorzystano testy statystyczne zaimplementowane w module `analize_results.py`.

H1: Porównanie modeli LLM oraz paradygmatów przeszukiwania

Hipoteza H1 zakładała, że wybór modelu językowego oraz paradygmatu przeszukiwania wpływa znacząco na jakość odpowiedzi. Zakładano, że model wnioskujący GPT-5.5 (C2) osiągnie najwyższą jakość merytoryczną wśród klasycznych podejść RAG (kosztem dłuższego czasu generowania). Ponadto sformułowano dwie podhipotezy: H1c (porównanie podejścia agentowego C7 z rurociągiem bazowym C1) i H1d (porównanie ultra-taniego chmurowego modelu C8 z lokalnym modelem C3).

- **Weryfikacja merytoryczna (H1a):** Wykonano powiązany test t-Studenta (paired t-test) porównujący oceny ogólne konfiguracji C1 (GPT-5.4) i C2 (GPT-5.5) na tej samej próbie 25 pytań. Uzyskano wartość statystyki testowej $t = 16,9706$ przy wartości $p = 7,178 \times 10^{-15}$. Ponieważ $p < 0,05$, różnicę uznano za skrajnie istotną statystycznie. Wielkość efektu mierzona współczynnikiem d Cohena wyniosła $d = 3,394$. Według klasyfikacji Cohena oznacza to gigantyczną wielkość efektu.
- **Weryfikacja wydajnościowa (H1b):** Średni czas odpowiedzi modelu GPT-5.5 (5,006 s) jest niemal dwukrotnie wyższy niż modelu GPT-5.4 (2,593 s). Test t-Studenta dla czasów odpowiedzi potwierdził istotność tej różnicy ($p < 0,001$).
- **Weryfikacja podejścia agentowego (H1c):** Porównano oceny jakościowe podejścia agentowego C7 (Claude Sonnet 4.6, śr. 4,928) i klasycznego rurociągu bazowego C1 (GPT-5.4, śr. 4,400). Test t-Studenta wykazał statystykę $t = 13,2665$ przy wartości $p = 1,524 \times 10^{-12}$. Oznacza to skrajną istotność statystyczną ($p < 0,05$) z gigantyczną wielkością efektu ($d = 2,653$). Podejście agentowe oparte na eksploracji plików zapewnia bezkonkurencyjną jakość, ugruntowanie faktyczne i brak halucynacji, aczkolwiek okupione jest to skrajnym wzrostem czasu odpowiedzi (średnio 25,079 s) oraz wysokim kosztem API.
- **Weryfikacja modelu budżetowego (H1d):** Porównano oceny ogólne ultra-taniego modelu chmurowego GPT-5-mini w konfiguracji C8 (śr. 4,072) z darmowym modelem lokalnym Llama 3.2 (3B) w konfiguracji C3 (śr. 3,968). Paired t-test wykazał wartość statystyki $t = 3,3754$ przy $p = 0,0025$ ($p < 0,05$). Wyznaczony współczynnik Cohena wyniósł $d = 0,675$, co oznacza średnio-silny, statystycznie istotny efekt. Model GPT-5-mini stanowi efektywniejszą alternatywę merytoryczną od lokalnego wariantu Llama 3.2 (3B).

Hipoteza **H1 została w pełni zweryfikowana pozytywnie.**

H2: Wymiarowość osadzeń

Hipoteza H2 zakładała, że wyższa wymiarowość osadzeń semantycznych (MPNet 768d vs MiniLM 384d) przełoży się na wzrost kompletności odpowiedzi. Porównano średnie oceny kryterium kompletności (ang. *Completeness*) dla konfiguracji C1 (MiniLM, śr. 4,120) oraz C4 (MPNet, śr. 4,720). Test t-Studenta dla prób zależnych wykazał wartość statystyki $t = 8,0483$ przy wartości $p = 2,836 \times 10^{-8}$. Różnica jest statystycznie wysoce istotna ($p < 0,05$). Wyznaczona wielkość efektu Cohena wyniosła $d = 1,610$, wskazując na bardzo silny wpływ wyboru modelu osadzeń na zdolność retrievera do znajdowania kompletnego kontekstu. Hipoteza **H2 została zweryfikowana pozytywnie.**

H3: Strategia dzielenia tekstu

Hipoteza H3 zakładała nieliniowy wpływ długości fragmentu na jakość systemu, wskazując rozmiar 500 słów (C1) jako optymalny kompromis. W celu zbadania istotności różnic między konfiguracjami C5 (250 słów, śr. ogólna 4,584), C1 (500 słów, śr. ogólna 4,400) oraz C6 (1000 słów, śr. ogólna 4,376) przeprowadzono jednoczynnikową analizę wariancji (one-way ANOVA). Uzyskano wartość statystyki

testowej $F = 9,9644$ przy wartości $p = 0,00015$. Różnica między strategiami segmentacji jest zatem statystycznie istotna ($p < 0,05$).

Szczegółowa analiza kryteriów cząstkowych wykazała, że:

- Konfiguracja o małych fragmentach C5 osiągnęła najwyższą ocenę zwiezłości (4,720) oraz jasności (4,800), uległa jednak pogorszeniu w wymiarze kompletności (4,000).
- Konfiguracja o dużych fragmentach C6 osiągnęła bardzo wysoką kompletność (4,880), uległa jednak degradacji w wymiarach jasności (4,000) i zwiezłości (4,000).
- Konfiguracja bazowa C1 (500 słów) oferuje najbardziej zrównoważony rozkład ocen (poprawność: 4,520, kompletność: 4,120, zwiezłość: 4,240). Wbrew założeniom hipotezy H3 to konfiguracja C5 (250 słów) uzyskała najwyższą średnią jakość ogólną (4,584) spośród tych trzech wariantów. Wynika to z faktu, że mniejszy kontekst sprzyjał precyzji generowania odpowiedzi przez model GPT-5.4, co ograniczyło nadmierną rozwlekłość tekstu.

Hipoteza **H3 została zweryfikowana częściowo pozytywnie**. Potwierdzono nieliniowy wpływ strategii segmentacji, jednak mniejsze fragmenty okazały się skuteczniejsze, niż pierwotnie zakładano.

H4: Opłacalność kosztowa

Hipoteza H4 zakładała, że płatne komercyjne modele językowe (C1, C2, C7, C8) oferują na tyle wysoką jakość, że w pełni uzasadniają koszty finansowe w stosunku do darmowych modeli lokalnych (C3). Wyznaczono koszt przypadający na pojedynczy punkt jakościowy (ang. *Cost per Quality Point*) wyliczony jako stosunek średniego kosztu zapytania do średniej oceny ogólnej jakości. Zestawienie wyników przedstawia Tabela 6.9.

Tabela 6.9: Koszt przypadający na pojedynczy punkt jakościowy (Cost per Quality Point) dla wybranych konfiguracji.

Konfig.	Śr. koszt zapytania [\$]	Jakość	Koszt / punkt jakości [\$]
C1 (GPT-5.4)	0,0003	4,400	0,000068
C2 (GPT-5.5)	0,0055	4,880	0,001127
C3 (Llama 3.2, 3B)	0,0000	3,968	0,000000
C7 (Agentic Claude)	0,1458	4,928	0,029586
C8 (GPT-5-mini)	0,0001	4,072	0,000025

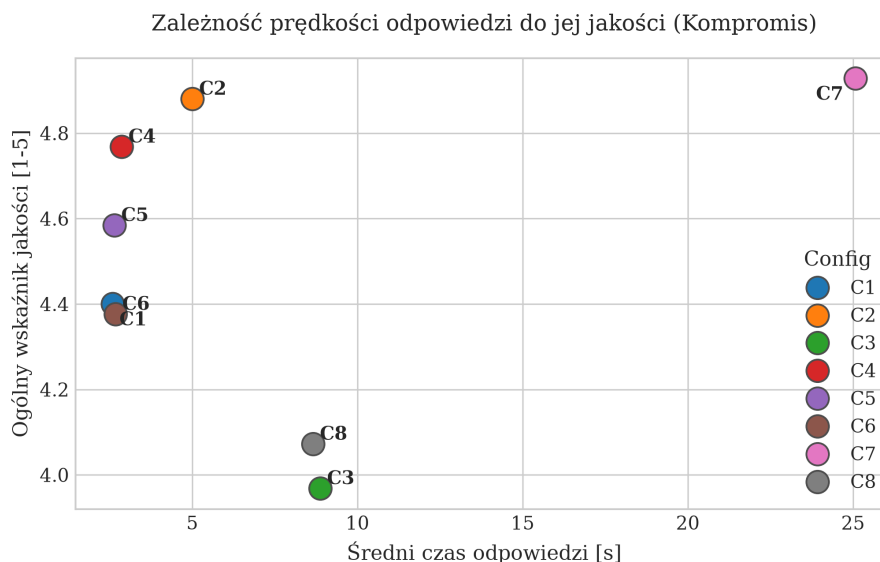
Niskie koszty operacyjne modelu GPT-5-mini (zaledwie 0,002\$ za cały zestaw 25 pytań) oraz jego przewaga jakościowa nad modelem lokalnym (4,072 vs 3,968) sprawiają, że model ten stanowi optimum ekonomiczne wśród wszystkich wariantów komercyjnych (koszt za punkt jakości na poziomie 0,000025\$). Model GPT-5.4 (C1) oferuje równie wysoki współczynnik opłacalności przy znacząco wyższej jakości ogólnej (4,400). W przypadku flagowego modelu GPT-5.5 (C2) koszt rośnie ponad siedemnastokrotnie przy wzroście jakości o jedynie 11% w stosunku do C1. Uwidacznia to malejące przychody krańcowe. Z kolei podejście agentowe C7 jest bardzo kosztowne operacyjnie (koszt punktu jakości na poziomie 0,029586\$ – około 1100-krotnie droższy od GPT-5-mini). Przez to jego stosowanie w celach komercyjnych jest uzasadnione wyłącznie w krytycznych zadaniach o najwyższym rygorze merytorycznym, gdzie budżet nie stanowi ograniczenia. Hipoteza **H4 została zweryfikowana pozytywnie**.

6.5.2 Analiza kompromisów (Trade-off Analysis)

Projektowanie systemów RAG wymaga od inżynierów oprogramowania ciągłego podejmowania decyzji kompromisowych. Zebrane dane pozwalają na precyzyjne skwantyfikowanie tych zależności.

Szybkość a jakość

Wzajemną zależność między czasem odpowiedzi a uzyskaną jakością przedstawiono na wykresie rozrzutu na rysunku 6.5.



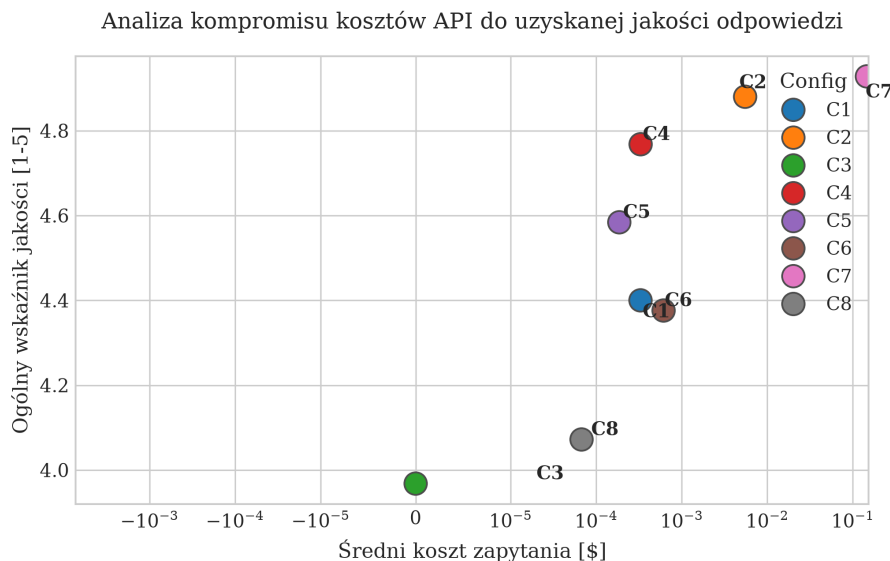
Rysunek 6.5: Analiza kompromisu między czasem odpowiedzi (szybkością) a ogólną jakością generowanych odpowiedzi. Źródło: opracowanie własne.

Rysunek 6.5 ilustruje, że optymalny punkt kompromisowy reprezentuje konfiguracja C4 (krótki czas odpowiedzi 2,866 s przy bardzo wysokiej jakości 4,768). Podejście agentowe C7 oferuje bezprecedensową, najwyższą jakość (4,928), lecz jest ono opóźnione przez wysoki czas oczekiwania (średnio 25,079 s). Ogranicza to jego zastosowanie w systemach interaktywnych działających w czasie rzeczywistym. Konfiguracja C2 (GPT-5.5) oferuje świetną jakość (4,880) przy znacznie rozsądniejszym opóźnieniu (5,006 s). Z kolei model chmurowy GPT-5-mini (C8) osiąga czas odpowiedzi 8,662 s ze stabilną jakością 4,072. Stanowi on bezpośredni odpowiednik wydajnościowy dla modelu lokalnego C3 (8,879 s), eliminując zarazem konieczność posiadania kosztownych zasobów sprzętowych.

Koszt a jakość

Kompromis między nakładami finansowymi a jakością odpowiedzi dla konfiguracji komercyjnych zaprezentowano na rysunku 6.6.

Rysunek 6.6 dowodzi istnienia wykładniczego wzrostu kosztów przy próbie osiągnięcia maksymalnych parametrów jakościowych przy użyciu standardowych i agentowych architektur chmurowych. Przejście z poziomu jakości 4,400 (C1) do poziomu 4,768 (C4) odbywa się bezkosztowo. Koszt zapytania pozostaje na poziomie 0,0003\$. Dalsze dążenie do maksymalizacji jakości za pomocą modeli wnioskujących w konfiguracji C2 (GPT-5.5) wiąże się ze znacznym skokiem kosztów do poziomu 0,0055\$ za zapytanie. Jednakże absolutnym ekstremum pod względem kosztowym jest podejście agentowe C7. Generuje ono średni koszt na poziomie aż 0,1458\$ za pojedyncze zapytanie. Pozycjonuje to ten wariant poza standardową skalą opłacalności masowych wdrożeń komercyjnych. Z drugiej strony model GPT-5-mini (C8) redefiniuje dolną granicę kosztów komercyjnych. Zapewnia on przyzwoitą jakość (4,072) przy niemal zerowym jednostkowym koszcie (0,0001\$), stanowiąc doskonałą alternatywę budżetową. Konfiguracja C4 stanowi pod tym względem najbardziej efektywnie ekonomicznie rozwiązanie typu *sweet-spot*.



Rysunek 6.6: Analiza kompromisu kosztów finansowych interfejsu API do uzyskanej jakości odpowiedzi. Źródło: opracowanie własne.

Rozmiar kontekstu a precyzja

Wpływ rozmiaru fragmentacji tekstu na poszczególne kryteria jakościowe zilustrowano na wykresie liniowym na rysunku 6.7.

Rysunek 6.7 obrazuje przeciwstawne wektory wpływu rozmiaru fragmentacji. Zwiększanie rozmiaru segmentu z 250 do 1000 słów podnosi ocenę kompletności odpowiedzi, ponieważ model otrzymuje bogatszy kontekst semantyczny. Jednocześnie powoduje to spadek poprawności merytorycznej wynikający z rozmycia istotnych informacji w szumie pobocznych zdań.

6.5.3 Ograniczenia i zagrożenia dla trafności (Threats to Validity)

Wiarygodność wniosków podlega zagrożeniom, które zminimalizowano poprzez:

- **Trafność wewnętrzną:** Zastosowanie ślepej próby (ocena odpowiedzi bez znajomości konfiguracji).
- **Trafność zewnętrzną:** Dobór różnorodnych tematycznie artykułów.
- **Trafność statystyczną:** Wykazanie silnych efektów ($d > 1,5$) przy próbie 25 pytań, co pozwoliło na uzyskanie istotności $p < 0,05$.

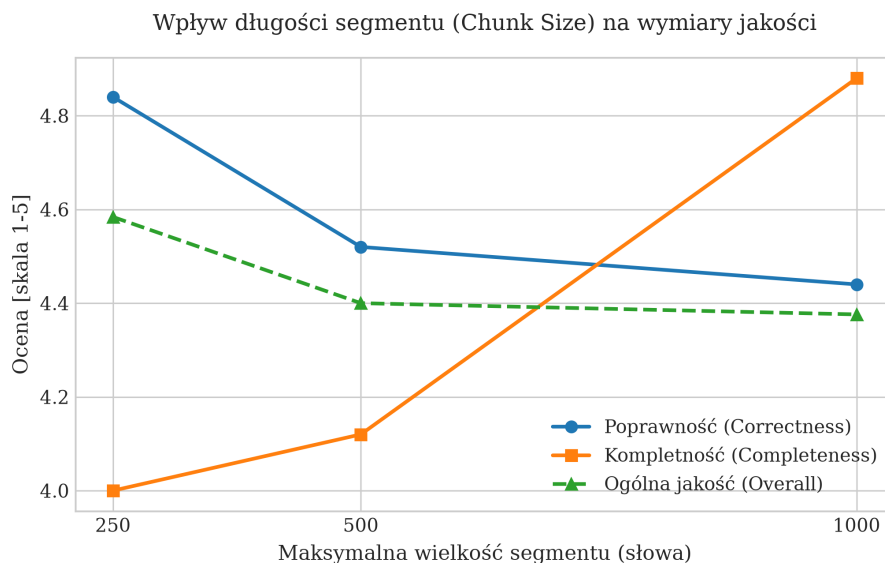
6.6 Podsumowanie rozdziału

Badania dostarczyły szczegółowych danych o operacyjnych i merytorycznych aspektach systemów generowania wspomaganego wyszukiwaniem.

6.6.1 Odpowiedzi na pytania badawcze

W oparciu o uzyskane wyniki sformułowano następujące odpowiedzi:

- **Odpowiedź na RQ1:** Wybór modelu LLM oraz paradygmatu wyszukiwania ma dominujący wpływ na czas i jakość odpowiedzi. Podejście agentowe C7 oferuje najwyższą jakość (4,928)



Rysunek 6.7: Wpływ maksymalnej wielkości segmentu tekstu (Chunk Size) na wymiary poprawności, kompletności oraz ogólnej jakości odpowiedzi. Źródło: opracowanie własne.

kosztem najwyższych opóźnień (25,079 s). Model wnioskujący GPT-5.5 (C2) zapewnia niemal zbliżoną jakość (4,880) przy znacznie niższym czasie (5,006 s). Zoptymalizowany model chmurowy GPT-5.4 (C1) oraz tani GPT-5-mini (C8) oferują bardzo dobre wskaźniki jakościowe (odpowiednio 4,400 oraz 4,072) przy krótkim czasie (odpowiednio 2,593 s oraz 8,662 s). Lokalny model Llama 3.2 (3B, C3) ustępuje pod względem jakości (3,968) i wykazuje wysokie opóźnienia (8,879 s) z uwagi na ograniczenia sprzętowe lokalnej inferencji.

- **Odpowiedź na RQ2:** Zastosowanie bardziej złożonego modelu osadzeń semantycznych MPNet (768d) w konfiguracji C4 przyniosło statystycznie wysoce istotny wzrost kompletności odpowiedzi (z 4,120 do 4,720, przy p-value na poziomie $2,8 \times 10^{-8}$), co potwierdza lepszą selekcję kontekstu przez wyżej wymiarowe wektory.
- **Odpowiedź na RQ3:** Optymalnym rozmiarem fragmentu tekstu okazało się 250 słów (C5). Wbrew teoretycznym założeniom o wyższości rozmiaru średniego (500 słów), mniejsze segmenty pozwoliły na wyeliminowanie szumu informacyjnego, co przełożyło się na wyższą precyzję syntezy i najwyższą ocenę ogólną (4,584) w grupie testów chunkingu.
- **Odpowiedź na RQ4:** Korzystanie z komercyjnych interfejsów API okazało się opłacalne. Pojawienie się modeli takich jak GPT-5-mini (C8) redefiniuje tę kalkulację. Przy koszcie rzędu 0,0001\$ za zapytanie, konfiguracja C8 oferuje wyższą jakość i brak halucynacji w stosunku do darmowego modelu lokalnego (C3), stanowiąc ekonomiczny punkt odniesienia. Zoptymalizowany GPT-5.4 (C1) zapewnia optymalny stosunek jakości do kosztów (0,0003\$ na zapytanie). Z kolei użycie modeli wnioskujących (C2) oraz ciężkich pętli agentowych (C7) generuje niewspółmiernie wyższe koszty i powinno być zarezerwowane wyłącznie dla zadań o najwyższym stopniu złożoności logicznej.

6.6.2 Rekomendacje praktyczne

Na podstawie wyników sformułowano rekomendacje wdrożeniowe:

1. **Zalecana konfiguracja produkcyjna:** Najlepszy stosunek jakości do wydajności i kosztów

oferuje konfiguracja oparta na modelu językowym gpt-5.4, modelu osadzeń semantycznych all-mpnet-base-v2 (768d) oraz rozmiarze fragmentacji na poziomie 250–500 słów.

2. **Ograniczenie modeli lokalnych:** Stosowanie modeli lokalnych (np. Llama3.2 o rozmiarze 3B) za pośrednictwem silnika Ollama na maszynach klienckich bez dedykowanych kart graficznych o dużej przepustowości VRAM powinno być unikane w systemach wymagających interakcji w czasie rzeczywistym z uwagi na wysokie opóźnienia czasowe (powyżej 8 sekund).
3. **Selektywne użycie modeli wnioskujących:** Modele klasy gpt-5.5 powinny być wpięte do potoku RAG w sposób hybrydowy (np. poprzez mechanizm rutowania zapytań), będąc wywoływanymi wyłącznie przy zapytaniach o wysokiej złożoności logicznej, co pozwoli na optymalizację kosztów budżetowych projektu.

6.6.3 Kierunki dalszych badań

Prezentowane badania otwierają interesujące perspektywy dla dalszych prac naukowych:

- **Dynamiczny chunking:** Zbadanie skuteczności algorytmów dynamicznego podziału tekstu (np. segmentacja oparta na analizie semantycznej granic akapitów lub zmian gęstości słów kluczowych) w porównaniu do sztywnego podziału na stałą liczbę słów.
- **Reranking:** Wprowadzenie do potoku wyszukiwania dodatkowego komponentu powtórnego szeregowania (np. Cohere Rerank lub wbudowanych modeli Cross-Encoder), co mogłoby dodatkowo podnieść relewantność bez konieczności stosowania ciężkich modeli osadzeń.
- **Ewaluacja automatyczna:** Porównanie ocen eksperckich z ocenami generowanymi automatycznie przez niezależny model sędziowski (metodologia LLM-as-a-Judge oparta na modelach GPT-4o lub Claude Sonnet 4.6) pod kątem zgodności i korelacji ocen.

Wypracowane wnioski oraz rekomendacje wdrożeniowe stanowią merytoryczne podsumowanie części badawczej i stanowią podstawę do sformułowania końcowych wniosków w rozdziale podsumowującym.

Rozdział 7

Podsumowanie i wnioski końcowe

W niniejszym rozdziale podsumowano całość zrealizowanych prac badawczych i programistycznych. Zweryfikowano stopień osiągnięcia celów. Analiza mocnych i słabych stron wdrożonego systemu pozwoliła sformułować kierunki dalszego rozwoju zaimplementowanej architektury.

7.1 Realizacja celów pracy

Zaprojektowanie, implementacja i ocena modułowego systemu analizy treści internetowych opartego na architekturze RAG (ang. *Retrieval-Augmented Generation*) stanowiły główny cel rozprawy. Osiągnięto go w pełni. Realizacja zadania objęła następujące etapy:

1. **Przegląd literatury i stanu sztuki** – krytyczna analiza literatury objęła obszary reprezentacji semantycznej tekstu, dużych modeli językowych oraz baz wektorowych. Stanowiło to teoretyczny fundament dalszych prac.
2. **Projekt i implementacja architektury RAG** – strukturę systemu oparto na bibliotece Haystack v2 i asynchronicznym interfejsie FastAPI. Zaimplementowano asynchroniczne indeksowanie witryn oraz synchroniczny potok generowania odpowiedzi z automatycznymi przypisami źródłowymi.
3. **Rozwiązanie problemów integracyjnych** – opracowano komponent MetadataCleaner. Rozwiązał on problem ograniczeń bazy ChromaDB w zakresie typów danych obsługiwanych w metadanych, co wyeliminowało błędy podczas zapisu i wyszukiwania semantycznego.
4. **Badania empiryczne** – zdefiniowano i przetestowano osiem konfiguracji eksperymentalnych (C1–C8), obejmujących zarówno klasyczny potok RAG, jak i wariant agentowy oraz model niskobudżetowy. Przeprowadzona ocena ilościowa i jakościowa dostarczyła statystycznie istotnych danych. Na ich podstawie określono rzeczywisty wpływ doboru modeli językowych, osadzeń oraz strategii fragmentacji tekstu na sprawność systemu.

7.2 Wady i zalety zaproponowanego rozwiązania

Zbudowany system łączy zalety użytkowe z technicznymi ograniczeniami. Ich analiza pozwala na obiektywną ocenę przydatności proponowanej architektury.

7.2.1 Zalety systemu

- **Modułowość i rozszerzalność** – architektura Haystack v2 umożliwia bezproblemową wymianę poszczególnych komponentów. Przykładowo zastąpienie ChromaDB bazą Qdrant lub wdrożenie innego modelu językowego nie wymaga modyfikacji pozostałej części kodu.
- **Elastyczność kosztowo-sprzętowa** – fabryka generatorów pozwala na pracę w chmurze lub lokalnie. W wariantcie chmurowym system korzysta z modeli OpenAI w kilku poziomach cenowych (gpt-5-mini, gpt-5.4 oraz gpt-5.5), w lokalnym zaś z modelu Llama 3.2 (3B) uruchamianego na platformie Ollama. Ułatwia to kompromis między wydatkami finansowymi, jakością odpowiedzi a wymogami prywatności.

- **Tryb agentowy o najwyższej wierności** – konfiguracja C7 oparta na modelu Claude Sonnet 4.6 samodzielnie eksploruje dokumenty za pomocą dedykowanych narzędzi, bez uprzedniego indeksowania wektorowego. W ocenie eksperckiej osiągnęła ona najwyższą jakość odpowiedzi (4,928 na 5,0) oraz brak halucynacji, co czyni ją przydatną w zadaniach o najwyższym rygorze merytorycznym.
- **Weryfikowalność i odporność na halucynacje** – system rzadko generuje błędne informacje. Wynika to z restrykcyjnego konstruowania instrukcji systemowych oraz precyzyjnego mapowania fragmentów wektorowych na tekst źródłowy. Wyjściowa odpowiedź opiera się wyłącznie na dostarczonym kontekście.
- **Efektywność oczyszczania danych** – dwuetapowy proces pobierania treści radzi sobie z większością witryn. Wykorzystanie BeautifulSoup4 do stron statycznych i biblioteki Playwright do aplikacji typu SPA (ang. *Single Page Application*) zapewnia poprawną ekstrakcję surowego tekstu.

7.2.2 Wady i ograniczenia systemu

- **Opóźnienia generowania w trybie lokalnym** – lokalna konfiguracja C3 drastycznie wydłuża czas oczekiwania na odpowiedź. Latencja przekracza tu 8 sekund. Modele chmurowe potrzebują niecałych 2 sekund, co na słabszych maszynach klienckich czyni tryb lokalny nieprzydatnym do interakcji w czasie rzeczywistym.
- **Wrażliwość na złożoną strukturę witryn** – nietypowe układy stron generują szum informacyjny. Rozbudowane menu, reklamy czy paywalle utrudniają ekstrakcję. Narzędzie Playwright jedynie ogranicza to zjawisko, przez co zbędny tekst trafia do bazy wektorowej i obniża precyzję wyszukiwania.
- **Brak automatycznej synchronizacji danych** – baza wektorowa ma charakter statyczny. System nie potrafi samodzielnie zidentyfikować zmian ani modyfikacji wprowadzonych w treściach witryn, które zostały już wcześniej przetworzone. Aktualizacja wiedzy wymaga ponownego, ręcznego uruchomienia potoku przez operatora.
- **Koszt i opóźnienia trybu agentowego** – najwyższa jakość konfiguracji C7 wiąże się ze średnim czasem odpowiedzi rzędu 25 sekund oraz kosztem około 0,15\$ za zapytanie. Wielokrotne przesyłanie pełnych dokumentów w pętli wnioskowania wyklucza ten tryb z masowych zastosowań interaktywnych.

7.3 Kierunki dalszych badań i plany na przyszłość

Zbudowany prototyp stanowi stabilną podstawę do dalszych prac badawczych. Wytypowano cztery kierunki rozwoju wdrożonej architektury:

1. **Wdrożenie technologii GraphRAG** – integracja z grafową bazą danych (np. Neo4j) pozwala na automatyczne budowanie grafów wiedzy. Ułatwia to realizację zapytań o charakterze globalnym, wymagających syntezy rozproszonych wątków (np. „jakie są wspólne mianowniki między wszystkimi zaindeksowanymi artykułami?”).
2. **Hybrydowe wyszukiwanie i reranking** – połączenie wyszukiwania semantycznego z algorytmem BM25 zwiększa precyzję znajdowania informacji. Dodatkowa warstwa rerankera (np. Cohere Rerank lub bge-reranker) pozwala lepiej uszeregować dokumenty przed przekazaniem ich do generatora.

3. **Autonomiczni agenci sieciowi** – dodanie komponentów agentowych pozwoli na samodzielne przeszukiwanie sieci przez system. Decyzje o indeksowaniu nowych zasobów w locie mogą zapadać automatycznie na podstawie analizy braków w lokalnej bazie wiedzy.
4. **Kwantyzacja i optymalizacja modeli lokalnych** – techniki kwantyzacji (np. GGUF, AWQ) zmniejszają wymagania sprzętowe. Zastosowanie dedykowanych silników wnioskowania, takich jak vLLM lub TensorRT-LLM, pozwala skrócić czas generowania odpowiedzi.

7.4 Wnioski końcowe

Zrealizowane testy empiryczne oraz proces wdrożenia systemu pozwoliły na sformułowanie wniosków końcowych:

- Architektura RAG efektywnie eliminuje ograniczenia modeli językowych. Ogranicza zjawisko halucynacji i dostarcza aktualną wiedzę z zewnętrznych źródeł.
- Modele komercyjne (GPT-5.4 oraz GPT-5.5) podnoszą użyteczność systemów analitycznych. Wariant GPT-5.5 precyzyjnie syntetyzuje i osadza odpowiedzi w kontekście. Z kolei model GPT-5.4 stanowi kompromis między jakością wyników a kosztami zapytań API.
- Lokalne modele (np. Llama 3.2 3B) generują poprawne merytorycznie teksty w języku polskim. Zapewniają one pełną prywatność. Stanowią realną alternatywę dla usług chmurowych, o ile użytkownik akceptuje wyższą latencję.
- Podejście agentowe (konfiguracja C7, model Claude Sonnet 4.6) zapewnia najwyższą jakość i wierność odpowiedzi, lecz jego wysoki czas reakcji oraz koszt uzasadniają stosowanie go wyłącznie w zadaniach o krytycznym znaczeniu merytorycznym.
- Tani model chmurowy GPT-5-mini (konfiguracja C8) przewyższa jakością lokalny model Llama 3.2 (3B) przy znikomym koszcie zapytania. Zmienia to dotychczasowy kompromis między rozwiązaniem lokalnym a chmurowym na korzyść niedrogiej chmury.
- Efektywność systemu RAG zależy w równym stopniu od generatora LLM, co od poprawności indeksowania i czyszczenia danych. Decydujący wpływ mają szczegóły techniczne. Semantyczny podział tekstu na fragmenty oraz bezbłędna obsługa struktur metadanych w bazie wektorowej bezpośrednio przekładają się na stabilność działania całego rozwiązania.

Wyniki eksperymentalne potwierdzają postawioną hipotezę. Połączenie modeli osadzeń, baz wektorowych i generatorów językowych w jedną architekturę pozwala na budowę sprawnych narzędzi analitycznych. Opracowany system efektywnie wspiera proces selekcji i syntezy informacji z rozproszonych źródeł sieciowych.

Bibliografia

- [1] Ashish Vaswani i in. “Attention Is All You Need”. W: *Advances in Neural Information Processing Systems (NeurIPS)*. T. 30. 2017, s. 5998–5608.
- [2] Jacob Devlin i in. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. W: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. T. 1. 2019, s. 4171–4186.
- [3] OpenAI. *Introducing GPT-5*. 2025. URL: <https://openai.com/blog> (term. wiz. 30. 11. 2025).
- [4] Anthropic. *Claude Sonnet 4.6: Best-in-class coding and AI agent capabilities*. 2025. URL: <https://www.anthropic.com/news> (term. wiz. 30. 11. 2025).
- [5] Google DeepMind. *Gemini 2.5 Pro: 1 million token context with native multimodal support*. 2025. URL: <https://deepmind.google> (term. wiz. 30. 11. 2025).
- [6] Meta AI. *The Llama 4 herd: The beginning of a new era of natively multimodal AI*. 2025. URL: <https://llama.meta.com> (term. wiz. 30. 11. 2025).
- [7] xAI. *Grok 4: Real-time knowledge access and advanced reasoning*. 2025. URL: <https://x.ai> (term. wiz. 30. 11. 2025).
- [8] Patrick Lewis i in. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. W: *Advances in Neural Information Processing Systems (NeurIPS)*. T. 33. 2020, s. 9459–9474.
- [9] John Rupert Firth. “A synopsis of linguistic theory, 1930-1955”. W: *Studies in Linguistic Analysis 1957* (1957), s. 1–32.
- [10] Gerard Salton i Christopher Buckley. “Term-weighting approaches in automatic text retrieval”. W: *Information Processing & Management* 24.5 (1988), s. 513–523.
- [11] Tomas Mikolov i in. “Efficient estimation of word representations in vector space”. W: *arXiv preprint arXiv:1301.3781* (2013).
- [12] Jeffrey Pennington, Richard Socher i Christopher D. Manning. “GloVe: Global vectors for word representation”. W: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, s. 1532–1543.
- [13] Armand Joulin i in. “Bag of tricks for efficient text classification”. W: *arXiv preprint arXiv:1607.01759* (2016).
- [14] Matthew E. Peters i in. “Deep contextualized word representations”. W: *arXiv preprint arXiv:1802.05365* (2018).
- [15] Dan Hendrycks i in. “Measuring Massive Multitask Language Understanding”. W: *arXiv preprint arXiv:2009.03300* (2020).
- [16] Alex Wang i in. “SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems”. W: *Advances in Neural Information Processing Systems (NeurIPS)*. T. 32. 2019, s. 3266–3280.
- [17] Yinhan Liu i in. “RoBERTa: A robustly optimized BERT pretraining approach”. W: *arXiv preprint arXiv:1907.11692* (2019).
- [18] Victor Sanh i in. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. W: *arXiv preprint arXiv:1910.01108* (2019).

- [19] Zhenzhong Lan i in. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. W: *arXiv preprint arXiv:1909.11942* (2019).
- [20] Anthropic. *Claude 4.1 Opus: Advanced reasoning for complex tasks*. 2025. URL: <https://www.anthropic.com/news> (term. wiz. 30. 11. 2025).
- [21] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri i in. “The Llama 3 Herd of Models”. W: *arXiv preprint arXiv:2407.21783* (2024).
- [22] Albert Q. Jiang i in. “Mixtral of Experts”. W: *arXiv preprint arXiv:2401.04088* (2024).
- [23] An Yang i in. “Qwen2 Technical Report”. W: *arXiv preprint arXiv:2407.10671* (2024).
- [24] DeepSeek-AI. “DeepSeek-V3 Technical Report”. W: *arXiv preprint arXiv:2412.19437* (2024).
- [25] Marah Abdin i in. “Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone”. W: *arXiv preprint arXiv:2404.14219* (2024).
- [26] Alec Radford i in. “Learning transferable visual models from natural language supervision”. W: *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 2021, s. 8748–8763.
- [27] Nils Reimers i Iryna Gurevych. “Sentence-BERT: Sentence embeddings using siamese BERT-networks”. W: *arXiv preprint arXiv:1908.10084* (2019).
- [28] Nandan Thakur i in. “BEIR: A heterogenous benchmark for zero-shot evaluation of information retrieval models”. W: *arXiv preprint arXiv:2104.08663* (2021).
- [29] Chroma. *Chroma: The AI-Native Open-Source Embedding Database*. 2024. URL: <https://www.trychroma.com> (term. wiz. 06. 07. 2026).
- [30] Pinecone Systems. *Pinecone: The Vector Database for Machine Learning Applications*. 2024. URL: <https://www.pinecone.io> (term. wiz. 06. 07. 2026).
- [31] Weaviate. *Weaviate: Open Source Vector Database*. 2024. URL: <https://weaviate.io> (term. wiz. 06. 07. 2026).
- [32] Jianguo Wang i in. “Milvus: A Purpose-Built Vector Data Management System”. W: *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*. 2021, s. 2614–2627.
- [33] Qdrant. *Qdrant: Vector Similarity Search Engine*. 2024. URL: <https://qdrant.tech> (term. wiz. 06. 07. 2026).
- [34] Jeff Johnson, Matthijs Douze i Hervé Jégou. “Billion-scale similarity search with GPUs”. W: *IEEE Transactions on Big Data*. T. 7. 3. 2019, s. 535–547.
- [35] Yury A. Malkov i Dmitry A. Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. W: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 42.4 (2018), s. 824–836.
- [36] Peng Xu i in. “Retrieval Meets Long Context Large Language Models”. W: *International Conference on Learning Representations (ICLR)* (2024).
- [37] Nelson F. Liu i in. “Lost in the Middle: How Language Models Use Long Contexts”. W: *arXiv preprint arXiv:2307.03172* (2023).
- [38] Yunfan Gao i in. “Retrieval-augmented generation for large language models: A survey”. W: *arXiv preprint arXiv:2312.10997* (2023).
- [39] Parth Sarthi i in. “RAPTOR: Recursive abstractive processing for tree-organized retrieval”. W: *arXiv preprint arXiv:2401.18059* (2024).

- [40] Kishore Papineni i in. “BLEU: a Method for Automatic Evaluation of Machine Translation”. W: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2002, s. 311–318.
- [41] Chin-Yew Lin. “ROUGE: A Package for Automatic Evaluation of Summaries”. W: *Text Summarization Branches Out (ACL Workshop)*. 2004, s. 74–81.
- [42] Shahul Es i in. “RAGAS: Automated Evaluation of Retrieval Augmented Generation”. W: *arXiv preprint arXiv:2309.15217* (2023).
- [43] TruEra. *TruLens: Evaluation and Tracking for LLM Experiments*. 2024. URL: <https://www.trulens.org> (term. wiz. 06.07.2026).
- [44] Sonal Gupta i Gail E. Kaiser. “A survey on web data extraction and integration”. W: *ACM Computing Surveys (CSUR)* 48.2 (2015), s. 1–35.
- [45] Alan R. Hevner i in. “Design science in information systems research”. W: *MIS Quarterly* 28.1 (2004), s. 75–105.
- [46] Tianyi Zhang i in. “BERTScore: Evaluating Text Generation with BERT”. W: *arXiv preprint arXiv:1904.09675* (2019).
- [47] Student. “The Probable Error of a Mean”. W: *Biometrika* 6.1 (1908), s. 1–25.
- [48] Frank Wilcoxon. “Individual Comparisons by Ranking Methods”. W: *Biometrics Bulletin* 1.6 (1945), s. 80–83.
- [49] Tolga Bolukbasi i in. “Man is to computer programmer as woman is to homemaker? Debiasing word embeddings”. W: *Advances in Neural Information Processing Systems (NeurIPS)* 29 (2016), s. 4349–4357.
- [50] Emily M. Bender i in. “On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?” W: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*. 2021, s. 610–623.
- [51] Pydantic Services Inc. *Pydantic: Data Validation Using Python Type Hints*. 2024. URL: <https://docs.pydantic.dev> (term. wiz. 06.07.2026).
- [52] deepset. *Haystack: LLM Orchestration Framework for Building Production-Ready Applications*. 2024. URL: <https://haystack.deepset.ai> (term. wiz. 06.07.2026).
- [53] Sebastián Ramírez. *FastAPI Documentation*. 2024. URL: <https://fastapi.tiangolo.com> (term. wiz. 06.07.2026).
- [54] Ollama. *Ollama: Get Up and Running with Large Language Models Locally*. 2024. URL: <https://ollama.com> (term. wiz. 06.07.2026).
- [55] BerriAI. *LiteLLM: Unified Interface for 100+ LLM APIs*. 2024. URL: <https://docs.litellm.ai> (term. wiz. 06.07.2026).
- [56] Python Software Foundation. *Requests: HTTP for Humans*. 2024. URL: <https://requests.readthedocs.io> (term. wiz. 06.07.2026).
- [57] Leonard Richardson. *Beautiful Soup Documentation*. 2024. URL: <https://www.crummy.com/software/BeautifulSoup> (term. wiz. 06.07.2026).
- [58] Microsoft. *Playwright: Reliable End-to-End Testing and Browser Automation*. 2024. URL: <https://playwright.dev> (term. wiz. 06.07.2026).
- [59] Theskumar. *python-dotenv: Read Key-Value Pairs from a .env File*. 2024. URL: <https://github.com/theskumar/python-dotenv> (term. wiz. 06.07.2026).

- [60] OpenAI. *tiktoken: A Fast BPE Tokeniser for Use with OpenAI's Models*. 2024. URL: <https://github.com/openai/tiktoken> (term. wiz. 06.07.2026).
- [61] Charles R. Harris i in. "Array Programming with NumPy". W: *Nature* 585.7825 (2020), s. 357–362.
- [62] Wikipedia. *Artificial Intelligence*. 2024. URL: https://en.wikipedia.org/wiki/Artificial_intelligence (term. wiz. 06.07.2026).
- [63] Wikipedia. *Climate Change*. 2024. URL: https://en.wikipedia.org/wiki/Climate_change (term. wiz. 06.07.2026).
- [64] Wikipedia. *World War II*. 2024. URL: https://en.wikipedia.org/wiki/World_War_II (term. wiz. 06.07.2026).
- [65] Wikipedia. *COVID-19*. 2024. URL: <https://en.wikipedia.org/wiki/COVID-19> (term. wiz. 06.07.2026).
- [66] Wikipedia. *Cryptocurrency*. 2024. URL: <https://en.wikipedia.org/wiki/Cryptocurrency> (term. wiz. 06.07.2026).
- [67] Benjamin S. Bloom i in. *Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook I: Cognitive Domain*. David McKay Company, 1956.

Dodatek A

Słownik użytej terminologii i skrótów

Dodatek zawiera zestawienie terminologii, pojęć i skrótów stanowiących podstawę teoretyczną oraz implementacyjną pracy. Służy uporządkowaniu aparatu pojęciowego. Każda pozycja podaje polskie określenie, jego powszechnie przyjęty w literaturze angielski odpowiednik oraz wyjaśnienie znaczenia. Pozycje uporządkowano alfabetycznie według polskiej nazwy, w jednej spójnej liście.

Baza wektorowa (*Vector Database*) – wyspecjalizowany system bazodanowy. Przeznaczony jest do składowania, indeksowania i szybkiej analizy porównawczej wielowymiarowych wektorów.

Dostrajanie (*Fine-tuning*) – adaptacja wcześniej wytrenowanego modelu do wąskich, wyspecjalizowanych zastosowań. Proces ten realizuje się na mniejszym, sprofilowanym zbiorze danych.

Duży model językowy (*Large Language Model, LLM*) – sieć neuronowa o wysokiej liczbie parametrów, liczonej zazwyczaj w miliardach. Trening na wielkich wolumenach tekstu pozwala jej na wnioskowanie oraz generowanie spójnych wypowiedzi.

Generowanie wspomagane wyszukiwaniem (*Retrieval-Augmented Generation, RAG*) – technika rozszerzania bazy wiedzy modelu językowego poprzez dynamiczne dostarczanie mu zewnętrznych dokumentów w trakcie generowania odpowiedzi. Pozwala to na uziemienie faktograficzne wypowiedzi i redukuje halucynacje.

Halucynacja (*Hallucination*) – generowanie przez model faktograficznie błędnych treści. Zachowują one przy tym pozory poprawności i spójności stylistycznej.

Inżynieria podpowiedzi (*Prompt Engineering*) – metodyka projektowania instrukcji wejściowych dla modeli językowych. Ma na celu precyzyjne sterowanie zachowaniem i generowanymi wynikami algorytmu.

Korpus (*Corpus*) – reprezentatywny, ustrukturyzowany zbiór tekstów w wersji cyfrowej. Stanowi bazę empiryczną przy trenowaniu i ewaluacji modeli językowych.

Lematyzacja (*Lemmatization*) – sprowadzanie wyrazów odmienionych do formy podstawowej. Wymaga to pełnej analizy morfologicznej oraz uwzględnienia kontekstu zdaniowego.

Mechanizm uwagi (*Attention Mechanism*) – algorytm ważący istotność poszczególnym elementom sekwencji wejściowej. Pozwala modelowi dynamicznie skupić się na najważniejszych fragmentach kontekstu podczas generowania odpowiedzi.

Odległość euklidesowa (*Euclidean Distance*) – geometryczna odległość w linii prostej między dwoma punktami w przestrzeni wielowymiarowej. Służy jako metryka niepodobieństwa wektorów.

Okno kontekstowe (*Context Window*) – limit danych wejściowych przetwarzanych jednorazowo przez model językowy. Wyrażany jest liczbą tokenów i wyznacza zakres pamięci krótkotrwałej algorytmu.

Osadzenie słowa (*Word Embedding*) – odwzorowanie wyrazu w postaci wektora liczbowego w przestrzeni wielowymiarowej. Bliskość geometryczna wektorów odpowiada tu ich pokrewieństwu znaczeniowemu.

- Osadzenie zdania (*Sentence Embedding*)** – wektor o stałej długości kodujący informację semantyczną dłuższego fragmentu tekstu. Obejmuje on zdania, akapity bądź całe dokumenty.
- Oznaczanie części mowy (*Part-of-Speech Tagging, POS*)** – algorytmiczne przypisywanie poszczególnym wyrazom ich kategorii gramatycznych. Analiza opiera się na kontekście składniowym zdania, wyznaczając m.in. rzeczowniki, czasowniki czy przymiotniki.
- Podobieństwo cosinusowe (*Cosine Similarity*)** – geometryczna miara bliskości wektorów. Wyznacza się ją jako cosinus kąta między nimi, uzyskując wartości z przedziału $[-1, 1]$.
- Podobieństwo semantyczne (*Semantic Similarity*)** – stopień bliskości znaczeniowej między tekstami. Pozostaje on niezależny od różnic w zapisie leksykalnym i składniowym.
- Podział na fragmenty (*Chunking*)** – dekompozycja długiego tekstu na mniejsze, spójne części. Ułatwia to poprawne generowanie osadzeń i późniejszą selekcję precyzyjnych informacji.
- Prze-ranking (*Reranking*)** – wtórna faza selekcji dokumentów. Mniejsza podgrupa najlepiej rokujących wyników jest w niej ponownie oceniana za pomocą dokładniejszego, bardziej złożonego modelu.
- Przetwarzanie języka naturalnego (*Natural Language Processing, NLP*)** – interdyscyplinarny obszar sztucznej inteligencji. Obejmuje komputerową analizę, zrozumienie oraz generowanie ludzkiej mowy i tekstu.
- Przybliżone wyszukiwanie najbliższych sąsiadów (*Approximate Nearest Neighbor, ANN*)** – algorytmy przyspieszające przeszukiwanie przestrzeni wektorowych. Optymalizują czas odpowiedzi kosztem dopuszczenia minimalnych odchyłeń od optymalnego wyniku.
- Reprezentacja gęsta (*Dense Representation*)** – wektor, którego wszystkie lub niemal wszystkie współrzędne mają wartości niezerowe. Ułatwia to kodowanie złożonych zależności semantycznych.
- Rozpoznawanie jednostek nazwanych (*Named Entity Recognition, NER*)** – automatyczne wyodrębnianie i kategoryzowanie w tekście określonych nazw własnych. Pozwala na identyfikację m.in. osób, organizacji, lokalizacji geograficznych czy ram czasowych.
- Semantyka dystrybucyjna (*Distributional Semantics*)** – koncepcja lingwistyczna zakładająca, że znaczenie wyrazu definiuje jego otoczenie tekstowe. Słowa o zbliżonych kontekstach zyskują podobne reprezentacje.
- Tokenizacja (*Tokenization*)** – podział ciągłego tekstu na mniejsze jednostki, zwane tokenami. Reprezentują one zazwyczaj słowa, ich fragmenty lub pojedyncze znaki interpunkcyjne.
- Transformer (*Transformer*)** – głęboka architektura sieciowa wykorzystująca mechanizm samouwagi. Wyeliminowała ona konieczność stosowania struktur rekurencyjnych w przetwarzaniu sekwencji.
- Uczenie wstępne (*Pre-training*)** – pierwszy, najbardziej wymagający obliczeniowo etap treningu na danych nieetykietowanych. W jego trakcie model przyswaja ogólną strukturę i statystykę języka.
- Wymiarowość (*Dimensionality*)** – liczba składowych w wektorze reprezentującym dane. Współczesne modele osadzeń operują zazwyczaj w przestrzeniach o rozmiarze od 384 do 4096.

Wyszukiwanie gęste (*Dense Retrieval*) – metoda selekcji informacji oparta na porównywaniu wektorów osadzeń. Umożliwia dopasowanie dokumentów na poziomie semantycznym, a nie tylko tekstowym.

Wyszukiwanie rzadkie (*Sparse Retrieval*) – tradycyjne wyszukiwanie tekstowe oparte na dopasowywaniu słów kluczowych i statystykach ich występowania. Typowym przykładem takiego podejścia jest algorytm BM25.

Spis rysunków

2.1	Uproszczona ilustracja liniowości przestrzeni wektorowej Word2Vec: dodanie różnicy wektorów „kobieta” i „mężczyzna” do wektora „król” daje wektor bliski wektorowi „królowa”.	6
2.2	Podobieństwo cosinusowe jako kąt θ między dwoma wektorami osadzeń; im mniejszy kąt, tym większe podobieństwo semantyczne.	15
2.3	Uproszczony schemat dwufazowej architektury systemu RAG: indeksowanie offline (górny potok) oraz wyszukiwanie i generowanie online (dolny potok)	17
3.1	Schemat blokowy ogólnej architektury modułowej proponowanego systemu.	24
3.2	Przebiegi przetwarzania danych w proponowanym systemie.	26
3.3	Etapy przetwarzania wstępnego w wejściowym potoku danych.	28
5.1	Schemat architektury warstwowej zaimplementowanego systemu analizy treści.	39
6.1	Średni czas indeksowania artykułów dla konfiguracji C1–C8. Źródło: opracowanie własne.	55
6.2	Rozkład czasu odpowiedzi na zapytania w konfiguracjach C1–C8. Źródło: opracowanie własne.	56
6.3	Rozbicie czasu odpowiedzi na składowe rurociągu RAG dla badanych konfiguracji. Źródło: opracowanie własne.	57
6.4	Wielowymiarowa ocena jakości odpowiedzi dla wybranych konfiguracji (C1, C2, C7, C8). Źródło: opracowanie własne.	59
6.5	Analiza kompromisu między czasem odpowiedzi (szybkością) a ogólną jakością generowanych odpowiedzi. Źródło: opracowanie własne.	63
6.6	Analiza kompromisu kosztów finansowych interfejsu API do uzyskanej jakości odpowiedzi. Źródło: opracowanie własne.	64
6.7	Wpływ maksymalnej wielkości segmentu tekstu (Chunk Size) na wymiary poprawności, kompletności oraz ogólnej jakości odpowiedzi. Źródło: opracowanie własne.	65

Spis tabel

2.1	Porównanie głównych metod reprezentacji tekstu	7
2.2	Porównanie wiodących dużych modeli językowych (stan na koniec 2025 r.)	9
2.3	Porównanie głównych algorytmów indeksowania wektorowego	14
3.1	Schemat struktury metadanych dla fragmentów tekstowych w bazie wektorowej.	27
3.2	Szczegółowa specyfikacja konfiguracji badawczych (C1–C8) zdefiniowanych w module testowym.	30
4.1	Porównanie frameworków webowych w języku Python pod kątem przydatności w systemach RAG.	35
4.2	Charakterystyka porównawcza wybranych modeli osadzeń tekstowych.	35
4.3	Porównanie systemów zarządzania bazami wektorowymi.	36
4.4	Zależności programistyczne i biblioteki pomocnicze systemu.	37
5.1	Stos technologiczny zaimplementowanego systemu analizy treści internetowych.	38
6.1	Konfiguracje testowe (C1–C8): modele składowe oraz cel badawczy.	49
6.2	Artykuły źródłowe wykorzystane w zbiorze testowym.	49
6.3	Pełna lista 25 pytań testowych użytych w ewaluacji, wraz z artykułem źródłowym i typem kognitywnym.	50
6.4	Wydajność procesu indeksowania artykułów dla konfiguracji C1–C8 (opis konfiguracji: C1 – baseline GPT-5.4; C2 – GPT-5.5; C3 – Llama 3.2 (3B) lokalnie; C4 – MPNet 768d; C5 – chunk 250 słów; C6 – chunk 1000 słów; C7 – agent Claude; C8 – GPT-5-mini; zob. też Tabelę 6.1).	55
6.5	Wydajność procesu udzielania odpowiedzi oraz koszty interfejsu API dla konfiguracji C1–C8.	56
6.6	Średni rozkład czasowy wykonania poszczególnych składowych potoku zapytań (w sekundach) dla konfiguracji C1–C8.	57
6.7	Średnie oceny kryteriów jakościowych odpowiedzi uzyskane w ramach oceny eksperckiej dla konfiguracji C1–C8.	58
6.8	Średnie oceny ogólnej jakości odpowiedzi w zależności od poziomu kognitywnego pytań dla konfiguracji C1–C8.	60
6.9	Koszt przypadający na pojedynczy punkt jakościowy (Cost per Quality Point) dla wybranych konfiguracji.	62

Spis listingów

2.1	Przykładowy szablon promptu integrujący pobrany kontekst z zapytaniem użytkownika .	18
5.1	Deklaratywna konfiguracja systemu oparta na bibliotece pydantic-settings (fragment klasy Settings).	40
5.2	Funkcja fabryki tworząca właściwy generator Haystack na podstawie konfiguracji backendu.	41
5.3	Kod źródłowy niestandardowego komponentu MetadataCleaner oczyszczającego metadane dla ChromaDB.	42
5.4	Konstrukcja potoku indeksowania w bibliotece Haystack: rejestracja komponentów i ich połączenie w graf przetwarzania.	43
5.5	Uproszczony punkt końcowy /ask uruchamiający potok zapytań RAG i zwracający odpowiedź wraz ze źródłami.	44
5.6	Fragment pliku docker-compose.yml z definicją usług bazy podręcznej, backendu oraz interfejsu użytkownika.	45