Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Systemy Internetowe Wspomagania Zarządzania

Wykorzystanie grafowej organizacji wiedzy do poprawy rezultatów RAG (Retrieval Augmented Generation) dla tekstów narracyjnych

Michał Podstawski
Numer albumu 322907

promotor
dr hab. inż. Mariusz Kamola

WARSZAWA 2025

Wykorzystanie grafowej organizacji wiedzy do poprawy rezultatów RAG (Retrieval Augmented Generation) dla tekstów narracyjnych

Streszczenie. Dziedzina dużych modeli językowych jest jedną z najszybciej rozwijających się dziedzin współczesnej informatyki. Równocześnie z samymi modelami rozwijają się także techniki i narzędzia pozwalające na ich skuteczniejsze wykorzystanie. Do tej kategorii należy Retrieval Augmented Generation, tj. Generacja Wzbogacona o Wyszukiwanie. Metoda ta pozwala na przekazanie modelowi różnego typu dokumentów jako kontekstu, dzięki czemu potrafi on generować trafne odpowiedzi ich dotyczące.

Przedstawiona praca miała za cel poprawę wyników Retrieval Augmented Generation dla tekstów narracyjnych. W tym celu wykorzystana została grafowa organizacja wiedzy pochodzącej ze źródłowych tekstów. Teksty źródłowe dzielone są na mniejsze fragmenty, stające się pierwszymi węzłami grafu. Następnie fragmenty te są na zasadzie podobieństwa semantycznego grupowane w klastry. Klastry takie uchwytują konkretną ideę lub znaczenie. Istotne było w tym procesie posłużenie się teorią literatury i wyodrębnienie konstytutywnych aspektów takiego typu tekstów. Zwrócenie uwagi na te aspekty pozwala lepiej strukturyzować dane wejściowe i zachowywać znaczenia całości tekstu oryginalnego. Z wykorzystaniem możliwości dużych modeli językowych tworzone są podsumowania treści w takich klastrach, a wygenerowane podsumowania stają się węzłami w kolejnej warstwie grafu. Po utworzeniu grafu staje się on źródłem wyszukiwań w Retrieval Augmented Generation, a wybrane węzły dostarczają kontekstu dla pytań kierowanych do dużego modelu językowego.

Przeprowadzone eksperymenty pozwoliły udowodnić, że zaproponowane podejście poprawia obecnie istniejące rozwiązania i pozwala generować odpowiedzi o wyższej jakości i zgodności z danymi źródłowymi. W trakcie eksperymentów oprócz badania ogólnej poprawności sprawdzono też i poddano analizie wpływ poszczególnych parametrów na działanie rozwiązania. Uzyskane wyniki potwierdzają skuteczne działanie opracowanej metody oraz mogą dać asumpt do dalszych badań.

Słowa kluczowe: duże modele językowe, RAG, grafy

Use of graph-based knowledge organization to improve the results of RAG (Retrieval Augmented Generation) for narrative texts

Abstract. The field of Large Language Models is one of the fastest developing areas of nowadays IT. In parallel with models themselves, there is also rapid development of techniques and tools that enable their more efficient usage. This category includes Retrieval Augmented Generation. This method allows to pass different documents to the model as context, which enables it to generate correct answers to questions related to them.

Presented study is aimed at improving Retrieval Augumented Generation for narrative texts. In order to achieve this, graph knowledge organization for knowledge sourced from the base documents is implemented. The documents are split into smaller chunks and become first nodes of the graph. Afterwards, the chunks are clustered based on their semantic similarity. The clusters capture concrete ideas and meanings. The use of the theory of literature and definition of crucial aspects of narrative texts is vital for the process. Focus on those aspects improves building input data structure and helps in preservation of meanings of the original text. The clusters are summarized using Large Language Models capabilities, and summarizations become nodes of the graph on a higher level. After the graph is created, it becomes a source of lookup for Retrieval Augmented Generation and selected nodes are the context for the formulated questions.

Conducted experiments confirmed that proposed solution works better than existing solutions and enables generation of answers that have both high quality and accuracy. Additionally, during the experiments, the impact of the crucial parameters on the solution was analyzed. Provided results allow for the efficient use of the method and can be a starting point for further research.

Keywords: Large Language Models, RAG, graphs

Spis treści

1. Wstęp 7
2. Wprowadzenie 9
2.1. Large Language Models (Duże modele językowe) 9
2.2. Retrieval Augmented Generation (Generacja Wzbogacona o Wyszukiwanie) 12
2.3. Wykorzystanie grafów w Retrieval Augmented Generation 22
2.3.1. Grafy wiedzy 22
2.3.2. Hierarchiczne struktury dokumentów źródłowych 27
2.3.3. RAPTOR 28
3. Opis eksperymentu 31
3.1. Cel i zakres pracy 31
3.2. Ogólna koncepcja usprawnień 31
3.3. Narzędzia wykorzystane do implementacji 34
3.4. Szczegóły implementacji 35
3.4.1. Metoda budowy grafu 35
3.4.2. Klasy odpowiedzialne za sumaryzację 36
3.4.3. Klasy odpowiedzialne za uszczegółowienia 38
4. Wyniki 40
4.1. Narzędzia i zestawy danych 40
4.2. Szczegóły implementacji benchmarków 41
4.3. Analiza wyników 43
4.3.1. Poprawność odpowiedzi 43
4.3.2. Poprawność semantyczna odpowiedzi 45
4.3.3. Wpływ liczby wykorzystanych rezultatów na poprawność odpowiedzi 46
4.3.4. Wpływ liczby tokenów na poprawność odpowiedzi 48
4.3.5. Wpływ poszczególnych aspektów na finalną odpowiedź 50
4.3.6. Wpływ ilości węzłów uszczegóławiających na jakość odpowiedzi . . . 54
4.3.7. Wpływ długości fragmentów i streszczeń na jakość odpowiedzi 56
5. Podsumowanie 58
Bibliografia 61
Spis rysunków 66
Spis tabel 67

1. Wstęp

Popularyzacja generatywnej sztucznej inteligencji (GenAI) jest tym trendem w świecie IT, który najbardziej naznaczył ostatnie miesiące. Dzięki zastosowaniu uczenia maszynowego na olbrzymich zbiorach danych udało się wytrenować modele, które potrafią rozwiązać szereg złożonych zadań związanych z tworzeniem treści, od generowania tekstów, poprzez dźwięk i obraz, aż po video. Krokiem milowym było opublikowanie przez firmę OpenAI dostępnego publicznie chata ChatGPT, opartego o modele z rodziny GPT [1]. Stanowi on dostępny dla wszystkich dowód, że sztuczna inteligencja tego typu, potrafiąca bardzo dokładnie i wiarygodnie odpowiadać na zadane w języku naturalnym pytania, może stanowić dla człowieka nieocenioną pomoc. Szybko okazało się, że pomoc taka nie ogranicza się do wsparcia podczas wyszukiwania potrzebnych informacji albo generacji prostych treści, ale daje się rozciągnąć na zastosowania profesjonalne, a nawet naukowe. Zakres branż i dziedzin nauki, w których wdrożono możliwość wykorzystania GenAI, wydaje się trudny do objęcia - od różnego systemów wsparcia klienta i asystentów ułatwiających dostęp do wiedzy w organizacjach, poprzez branże sportową, gamingową i telewizyjną, aż po liczne zastosowania w biotechnologii i medycynie [2].

Upowszechnienie się modeli tego typu otworzyło jednak cały obszar nowych problemów do rozwiązania oraz możliwości do eksploracji. Zaczynają się one on wyzwań związanych ze sprzętem, gdyż zarówno trenowanie tak dużych modeli, jak i inferencja z ich użyciem, wymagają ogromnych mocy obliczeniowych. Ciągnie to za sobą błyskawiczny rozwój związany z dostarczaniem coraz to nowych rozwiązań technologicznych, szczególnie związanych z półprzewodnikami i GPU. Inną rozwijającą się kategorią są same modele, ich architektury, oraz architektury łączące różne modele w bardziej złożone całości. W ostatnim czasie pojawił się szereg usprawnień w tej dziedzinie, w tym kolejne wersje najpopularniejszych modeli, oraz ogromna ilość nowych modeli, które często znacznie przewyższają dotychczas istniejące. Dobrym świadectwem rozwoju są coraz lepsze wyniki osiągane w benchmarkach przez następne generacje rozwiązań. Rezultaty są często publikowane w dostępnej formie rankingów, takich jak Massive Text Embedding Benchmark (MTEB) [3], LMSYS Chatbot Arena [4] czy Open LLM Leaderboard [5].

Jeszcze innym obszarem rozwoju są prace związane z różnego typu narzędziami pomocniczymi i frameworkami, które pozwoliłyby lepiej wykorzystać istniejące modele. W zakresie modeli generujących tekst wysiłki koncentrują się przede wszystkim na dwóch kierunkach. Pierwszy to inżynieria zapytań (*prompt engineering*), tj. opracowanie takiej formy kierowanych do modelu zapytań, by uzyskać jak najlepsze odpowiedzi. Systemy tego typu mogą być bardzo złożone, konstruując łańcuchy, drzewa i grafy wnioskowań ([6], [7], [8]), dzięki którym rosną możliwości poprawnej odpowiedzi. Drugi nurt skupia się na inżynierii informacji, tj. na takim opracowaniu danych wprowadzanych wraz z zapytaniem do modelu, żeby zwiększyć ich trafność w określonym kontekście. Również tutaj pojawił się cały wachlarz możliwości, od dostosowania klasycznych relacyjnych baz danych dla

potrzeb związanych z użyciem dużych modeli językowych, aż po wielostopniowe systemy porządkowania informacji, by model zawsze mógł otrzymać najbardziej pasującą.

Prezentowana praca stanowi próbę rozszerzenia możliwości modeli językowych w dziedzinie operowania na dłuższych tekstach narracyjnych właśnie dzięki wprowadzeniu nowego sposobu strukturyzowania danych wejściowych. Opracowana metoda opiera się o grafową organizację wiedzy i pozwala na uzyskanie wyników przewyższających dostępne obecnie rozwiązania. Jako taka może być wykorzystana do budowy systemów posługujących się takimi danymi oraz stanowić impuls do dalszego rozwoju w tej dziedzinie.

2. Wprowadzenie

2.1. Large Language Models (Duże modele językowe)

Duże modele to najnowszy etap rozwoju uczenia maszynowego i sztucznej inteligencji związanych z przetwarzaniem języka naturalnego (*NLP - Natural Language Processing*). Ich architektury obejmują miliony parametrów, a źródłem do treningu stały się zbiory danych tekstowych o olbrzymim rozmiarze. W praktyce duże modele językowe są w stanie wybrać kolejne słowo o najwyższym prawdopodobieństwie dla zadanej sekwencji (stopień odchylenia od największego prawdopodobieństwa można regulować za pomocą tzw. *temperatury*); proces powtarzany jest iteracyjnie, aż do momentu, kiedy w sekwencji pojawi się token oznaczający zakończenie. Fundamentem tego rozwiązania jest zaproponowana w pracy *Attention is All You Need* [9] architektura tzw. transformera. Transformer jest modelem implementującym mechanizm uwagi własnej (*self-attention*), co pozwala na uchwycenie wielowymiarowych zależności występujących w tekstach, niezależnie od odległości pomiędzy konkretnymi słowami.

Jak wspomniano, duże modele językowe wywodzą się z technik przetwarzania języka naturalnego. Ewoluowały one od metod statystycznych (z użyciem n-gramów i podejścia probabilistycznego) do zaawansowanych metod uczenia maszynowego. Głównym ograniczeniem tych podejść było ograniczenie do rozpoznawania zależności w dłuższym tekście, co prowadziło do utraty jakości i problemów z analizą w bardziej złożonych zastosowaniach.

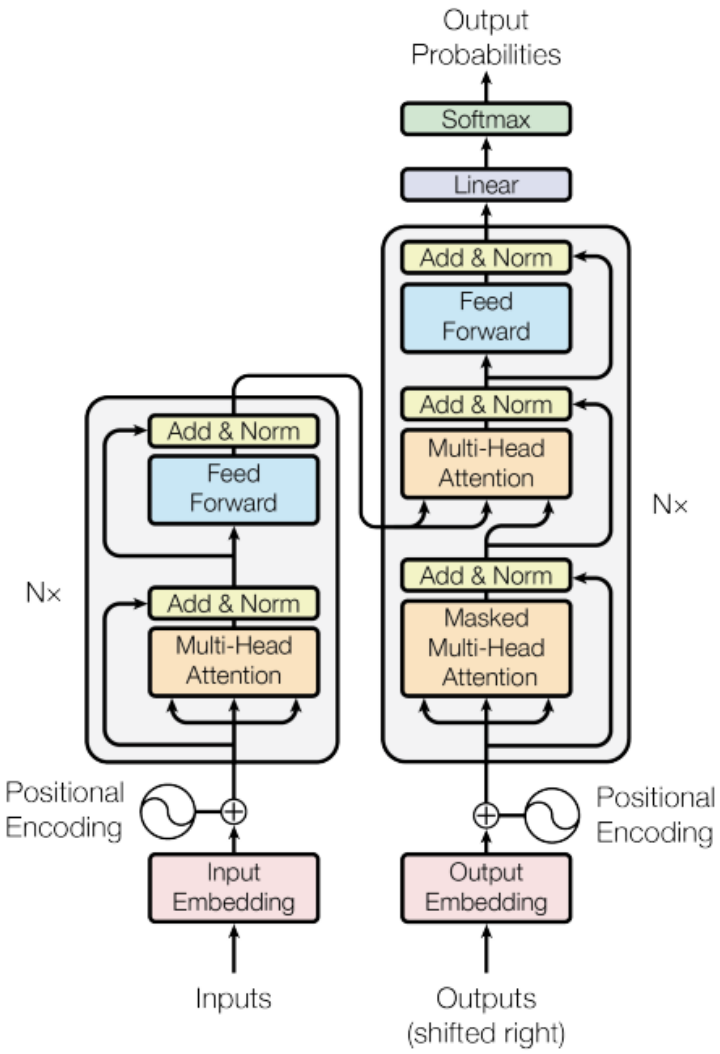
Problemy te znalazły częściowe rozwiązanie dzięki rozwojowi uczenia głębokiego i stworzeniu architektury sieci rekurencyjnych (RNN - *Recurrent Neural Network*). Sieci te potrafią lepiej modelować zależności w dłuższych danych sekwencyjnych, obarczone są jednak problemem zanikającego gradientu, a przez to tracą kontekst dla długich tekstów. Kolejnym krokiem było zaprojektowanie architektury LSTM (*Long short-term memory*). LSTM wykorzystuje komórki pamięci i pozwala lepiej sterować przepływem informacji, co pozwala ograniczyć wpływ zanikającego gradientu. Mimo wyraźnego postępu, wykrywanie zależności pomiędzy wyrazami o większych odległościach wciąż jednak bywa w nich problematyczne. Wyzwania te udało się rozwiązać dopiero dzięki opracowaniu wspomnianej już architektury transformer, która w nieco odmienny sposób podchodzi do problemu.

Architektura ta oparta jest o dwa główne komponenty: blok enkodera i blok dekodera. Enkoder analizuje sekwencje wejściową, i na podstawie tych danych dekoder generuje sekwencję wyjściową:

1. **Enkoder** składa się z kilku warstw, z których każda zawiera:
    - Komponent uwagi własnej (*self-attention*), która pozwala modelowi na analizę zależności między tokenami.

- Komponent sieci w pełni połączonej (*feed-forward*), która dokonuje nieliniowego przekształcenia danych.

2. **Dekoder** działa analogicznie do enkodera. Kluczową różnicą jest zastosowanie maskowanej uwagi własnej (*masked self-attention*). Rozwiązanie to działa podobnie do zwykłej uwagi własnej, ale w tym wypadku kolejne tokeny są maskowane (tj. ukryte) przed modelem. Wymusza to wygenerowanie kolejnego tokenu zgodnie z prawdopodobieństwem.



Rysunek 2.1. Architektura transformera. Ilustracja za [9].

Jednym z najistotniejszych komponentów transformera jest wspomniany mechanizm uwagi własnej (*self-attention*). Dzięki jego wykorzystaniu nie trzeba przetwarzać danych sekwencyjnie, ale można tego dokonać za jednym razem dla całości analizowanego tekstu. Schemat działania tego mechanizmu można przedstawić następująco:

1. Dla każdego tokenu w podanej sekwencji obliczana jest reprezentacja wektorowa, uwzględniająca jego kontekst w całości tekstu.

2. Następnie dla tokenów obliczane są trzy wektory: zapytanie (*query*), klucz (*key*) i wartość (*value*).
3. W kolejnym kroku obliczana jest uwaga dla każdego tokenu w stosunku do innych tokenów. Jest do tego wykorzystywany iloczyn skalarny zapytania i klucza. Wynik ten jest normalizowany przy użyciu funkcji softmax, co pozwala na przydzielenie większej uwagi ważnym tokenom.
4. Finalnie wektory wartości są skalowane przez obliczoną uwagę, co pozwala modelowi na przetworzenie najbardziej istotnych informacji z każdego tokenu.

W wypadku transformera używa się dla tego mechanizmu wielu równoległych głowic (*multi-head attention*) - tj. mechanizm uwagi stosowany jest wielokrotnie, ale z różnymi wartościami zapytania, klucza i wartości. Pozwala to na uchwycenie różnych zależności, na przykład jedna głowica może skupiać się głównie na krótkodystansowych zależnościach, a inna na bardziej odległych. Ostateczny wynik jest wynikiem konkatenacji rezultatów i przekształcenia za pomocą warstwy liniowej w jeden wektor wyjściowy.

Do zalet architektury transformera należą:

- **Efektywność w obsłudze długich sekwencji**: Dzięki uwadze własnej model jest w stanie dużo lepiej wychwytywać zależności, szczególnie pomiędzy odległymi od siebie tokenami.
- **Równoległość**: Mechanizm uwagi własnej pozwala na równoczesne przetwarzanie wszystkich tokenów w sekwencji. Dzięki temu czas trenowania modeli jest krótszy
- **Skalowalność**: Zaletą jest też skalowalność rozwiązania. W architekturze tranformera bezproblemowe jest dokładanie kolejnych warstw i parametrów, dzięki czemu szybko uzyskiwane są modele o coraz większych możliwościach.

Skalowalność sprawia, że wdrożone rozwiązania składają się z bardzo dużej ilości parametrów. Dla przykładu, już GPT-3 posiadał 175 miliardów parametrów [10]. Finalnym krokiem dla modeli zbudowanych w oparciu o opisaną architekturę jest trening na dużych ilościach danych. Dzięki temu uczą się one zależności występujących najczęściej pomiędzy tokenami i są w stanie przewidywać kolejne tokeny w nieznanych sekwencjach.

Ważnym momentem w rozwoju dużych modeli językowych było ich upublicznienie dla większego grona odbiorców. Dokonała tego firma OpenAI przez uruchomienie aplikacji ChatGPT, pozwalającej na prowadzenie konwersacji z dużym modelem językowym (początkowo GPT-3.5, potem GPT-4, GPT-4o i nowsze) [11]. Łatwość obsługi oraz szeroki zakres zadań, w których chat mógł być pomocny - od generacji tekstów, poprzez pomoc w rozwiązywaniu zadań technicznych (takich jak opracowanie gotowych fragmentów kodu źródłowego), aż po wsparcie w wyszukiwaniu informacji - stały się przyczyną olbrzymiego sukcesu. Ilość użytkowników przekroczyła 100 milionów, dzięki czemu ChatGPT stał się najszybciej rozwijającą się aplikacją w historii [12]. Wpłynęło to na wzrost powszechnej świadomości możliwości otwieranych przez wykorzystanie modeli językowych.

Zakres współczesnych zastosowań dużych modeli językowych jest bardzo szeroki:

- **Odpowiadanie na pytania**. Jedna z najbardziej rozpowszechnionych funkcjonalności dużych modeli językowych, polegająca na możliwości pytania modelu w języku naturalnym o potrzebne informacje, spopularyzowana głównie przez ChatGPT. Kategoria ta jest jednak dużo szersza, i coraz częściej obejmuje też różnego rodzaju wirtualnych asystentów, którzy pozwalają na zadawanie pytań dotyczących konkretnych dziedzin. Asystenci tacy wykorzystywani są do usprawnienia pracy z dokumentacją techniczną, umożliwiają pytanie o dokumenty związane z działalnością przedsiębiorstwa, mogą też zastępować człowieka jako pierwsza linia wsparcia i obsługi klienta.
- **Generowanie tekstu**. Pokrewną kategorią jest możliwość generowania tekstu. Duże modele mogą tworzyć teksty na zadany temat, a nawet w określonej konwencji. Możliwość ta często wykorzystywana jest do generacji zawartości stron internetowych, a także może być traktowana jako wsparcie w marketingu i dziennikarstwie.
- **Tłumaczenie tekstu**. Duże modele językowe są również bardzo skuteczne w tłumaczeniu zadanego tekstu. Dzięki opisanej wcześniej architekturze modele są w stanie dostarczać równoważne teksty w różnych językach, z zachowaniem sensu i poprawności gramatycznej.
- **Streszczanie tekstu**. Dodatkową funkcjonalnością oferowaną przez duże modele językowe jest zdolność streszczania tekstów. Zastosowanie to ma szczególne znaczenie w branżach, gdzie ważna jest szybka analiza danych i zdolność do szybkiego przeglądania dokumentów z różnych źródeł.
- **Wspomaganie programowania**. Istotnym zastosowaniem dużych modeli językowych jest wsparcie programowania. Modele takie trenowane są na dużej ilości kodu i dzięki temu potrafią zaproponować rozwiązania konkretnych problemów na podstawie kontekstu lub opisu dostarczonego w języku naturalnym. Przykładem narzędzia tej kategorii jest GitHub Copilot [13].

Pojawienie się i upowszechnienie dużych modeli językowych ma olbrzymi wpływ na obecny stan informatyki. Z jednej strony wciąż trwa dynamiczny rozwój technologii związanych z budową modeli, zwiększeniem ich możliwości i zakresu zastosowań. Z drugiej, dzięki szybko postępującemu rozwojowi narzędzi inżynierskich umożliwiających ich łatwą adaptację i wykorzystanie, z dnia na dzień rośnie liczba aplikacji dostępnych dla szerokiego grona użytkowników.

2.2. Retrieval Augmented Generation (Generacja Wzbogacona o Wyszukiwanie)

Pomimo swoich możliwości technologia LLM napotyka też liczne ograniczenia. Do najpoważniejszych należy fakt, że model trenowany jest na określonym, zamkniętym zbiorze danych, dla którego górnym limitem czasowym jest moment rozpoczęcia treningu. Z tego względu model nie potrafi udzielić poprawnych odpowiedzi na pytania o zdarzenia, które zaszły po zamknięciu zbioru.

Nie jest to jedyny problem. Operowanie na zamkniętych zbiorach danych jest przeszkodą we wdrożeniu dużych modeli językowych do bardziej szczegółowych zastosowań, gdyż specjalistyczna wiedza może być niewystarczająco reprezentowana w źródłowym zbiorze. Jeszcze poważniej problem ten rysuje się w przypadkach, w których pożądane jest wykorzystanie konkretnego, zazwyczaj niepublicznego korpusu wiedzy. Przykładem są coraz częściej wdrażane przez firmy systemy typu chatbot - *inteligentny asystent*, które, wykorzystując dokumenty przedsiębiorstwa, mogą udzielać odpowiedzi na pytania z nimi związane, na użytek wewnątrz organizacji lub dla jej klientów. Jasnym jest, że wszystkie tego typu dane, by mogły zastać użyte, należy osobno dostarczyć do modelu.

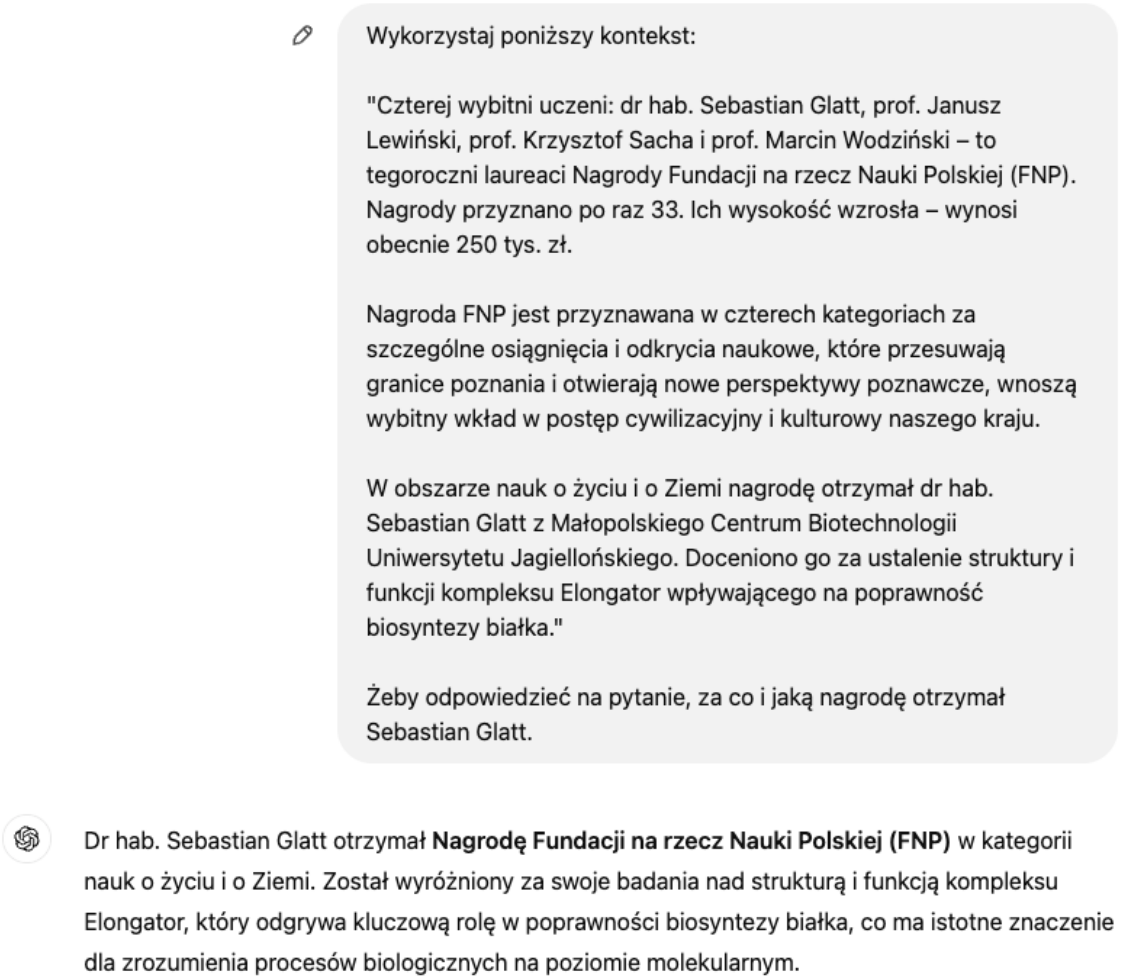
Pośrednio powiązane z tymi ograniczeniami jest zjawisko tzw. *halucynacji* (choć być może trafniej byłoby je opisać jako "zjawisko konfabulacji"), kiedy model co prawda udziela odpowiedzi, ale odpowiedź niezgodna jest ze stanem rzeczywistym, tj. kiedy model zmyśla odpowiedzi niepoparte faktyczną wiedzą. Może to być powodowane brakiem lub niedoborem odpowiednich informacji w zbiorze treningowym, lub przeciwnie, nadmierną ilością wzajemnie sprzecznych informacji.

Tradycyjnie w AI/ML adresuje się podobne potrzeby za pomocą *fine-tuningu* modelu, tj. dotrenowaniu modelu na dodatkowym zestawie danych, pochodzącym z domeny, w której model ma być wykorzystywany. Technika ta sprawdza się również w wypadku LLM ([14], [15]). *Fine-tuning* nie jest jednak pozbawiony wad. Podstawową jest fakt, że trening wymaga dostępu do odpowiednich zasobów obliczeniowych, co wiąże się ze sporymi kosztami. Inną wadą jest potrzebny na taki trening czas oraz konieczność posiadania eksperckiej wiedzy. Kolejnym problemem jest konieczność dostarczenia dużego i właściwie przygotowanego zestawu danych treningowych. Jeszcze trudniejszy jest przypadek, gdy oczekiwany korpus dodatkowej wiedzy może ulegać zmianom, bo każda taka zmiana wymaga dodatkowego treningu. Względy te sprawiają, że *fine-tuning* nie zawsze jest dającą się zastosować możliwością.

Duże modele językowe otwierają jednak dodatkowe opcje. Najważniejsza z nich wynika ze zdolności LLM do wykorzystania dodatkowych informacji umieszczonych w zapytaniu, nazywana *In-context learning (ICL)*, tj. uczenie się z kontekstu. Jedynym z najprostszych zastosowań *ICL* jest tak zwane "pytanie z kilkoma przykładami"(*few-shot prompting*). Może ono być wykorzystywane, żeby podać modelowi językowemu przykłady pytań i odpowiedzi zgodnie z oczekiwaną strukturą i szczegółowością. Dzięki podaniu takich przykładów, model potrafi lepiej odpowiedzieć na pytanie oraz ująć je w zadaną strukturę. Działanie można prześledzić na poniższym przykładzie:

Rysunek 2.2. *Few-shot prompting* z podaniem przykładowych pytań

In-context learning nie ogranicza się do możliwości podawania oczekiwanej struktury odpowiedzi. Zapytanie może także zostać wzbogacone o dodatkowy kontekst, który pozwala na udzielenie odpowiedzi na pytanie. Kontekst taki zawiera dodatkową wiedzę, która następnie jest wykorzystywana w generowanych komunikatach. Właśnie ta możliwość często pozwala zastąpić *fine-tuning* modelu. Z wykorzystaniem *in-context learningu* do projektowanego zapytania wystarczy dodać odpowiednie informacje, aby uzyskać odpowiedzi z nimi związane. Dla przykładu, jeśli celem jest uzyskanie odpowiedzi na pytania dotyczące najnowszych wydarzeń na świecie, wystarczy do zapytania dodać artykuł na ten temat. Model jest w stanie przetworzyć te informacje i udzielić aktualnych wyjaśnień. Co więcej, podawanie odpowiedniego kontekstu zmniejsza również skłonność modelu do *halucynacji* [16]. Zastosowanie takie można zobaczyć na poniższym przykładzie (podano fakt z dnia 30 października za Dziennikiem Naukowym [17] - więc fakt, który zaszedł po zamknięciu zbioru treningowego dla modelu):



Rysunek 2.3. *In-context learning* - przykład odpowiedzi na pytanie o fakty, które zaszły po zamknięciu zbioru treningowego dla modelu (zrzut ekranu z [1])

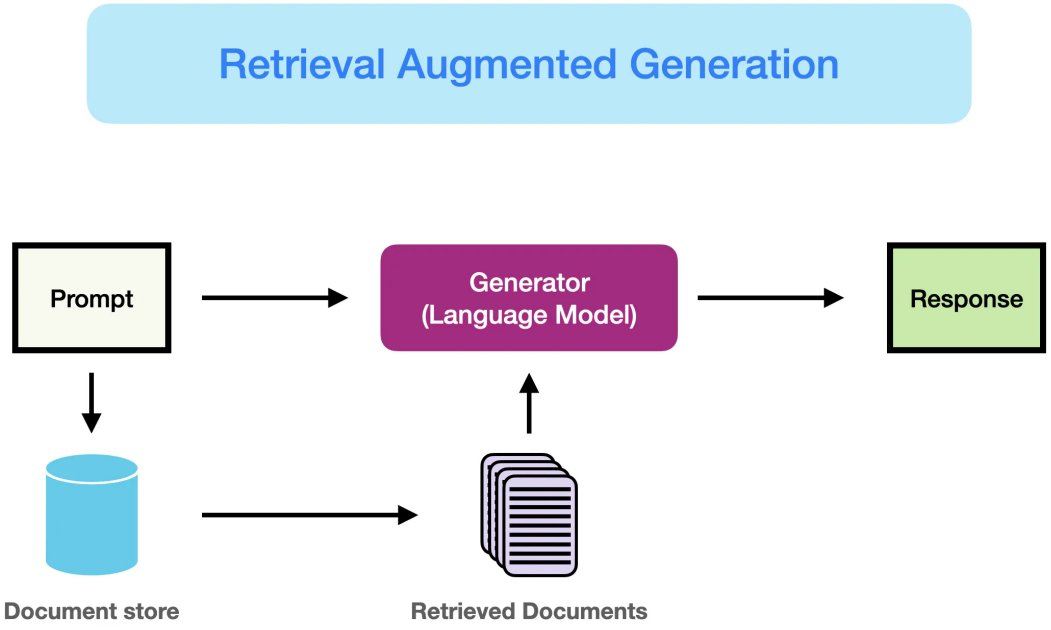
Podejście to trafia jednak na ograniczenia, związane przede wszystkim z długością kontekstu, który może być przekazany do modelu. Dla przykładu, model GPT-3.5 ma limit 16000 tokenów. To znaczny postęp względem poprzednich modeli (pierwsze generacje GPT-3.5 posiadały długość kontekstu 4000) [10], ale i tak w wielu wypadkach długość ta nie jest wystarczająca, szczególnie w wypadku operowania na bardzo obszernych dokumentach, albo gdy do odpowiedzi na pytanie konieczne jest skompilowanie informacji pochodzących w wielu rożnych źródeł.

Nie jest to jedynym problemem. Udowodniono również, że w wypadku długich dokumentów duże modele językowe mają tendencję do skupienia na informacjach znajdujących się na początku i na końcu podanego kontekstu, a pomijania tych, które znajdują się w środku. Zjawisku temu nadano nazwę "efektu zagubienia w środku"(*lost in the middle*) [18]. Rozwiązaniem tego ograniczenia jest podawanie informacji w bardziej zwartej formie, łatwiejszej do przyjęcia i zrozumienia przez model.

Innym problemem jest obciążenie obliczeniowe związane z przetwarzaniem bardzo

dużej ilości tokenów oraz związany z tym koszt. W przypadku podawania bardzo długich kontekstów model za każdym razem musi przetworzyć wszystkie tokeny, co wymaga dużych nakładów związanych z odpowiednim sprzętem i czasem. Również ta perspektywa wskazuje, że korzystniejszym jest ograniczenie informacji przekazywanych do kontekstu do tych faktycznie potrzebnych.

W związku z powyższymi problemami zaproponowano architekturę *Retrieval Augmented Generation (RAG)* - tj. Generacja Wzbogacona o Wyszukiwanie [16]. Stoi za nią idea, żeby dodatkową wiedzę potrzebną do odpowiedzi na pytanie wstrzykiwać do zapytania, ale żeby ograniczyć ją tylko do istotnych w kontekście odpowiedzi fragmentów. W najprostszej postaci rozwiązanie takie wygląda następująco. Ze względu na ograniczoną długość pytań dopuszczalnych przez modele tekst źródłowy jest dzielony na mniejsze fragmenty, tzw. *chunki* (ich wielkość zależna jest od architektury rozwiązania). Następnie sprawdzane jest podobieństwo wprowadzonego pytania i poszczególnych fragmentów. Z całego zestawu wyszukiwana jest określona ilość takich fragmentów, które najbardziej odpowiadają zadanemu pytaniu (*top k*). Stają się one kontekstem, który dostaje dodany do pytania podczas przesłania do modelu, tym samym zwiększając prawdopodobieństwo poprawnej odpowiedzi na zadane pytanie.



Rysunek 2.4. Schemat frameworka RAG (źródło: [19])

Do tradycyjnych metod wykorzystywanych do porównania podobieństwa pomiędzy pytaniem i *chunkami* należą przede wszystkim algorytmy TD-IDF i Okapi BM25. Są to algorytmy oceny tekstu używane w wyszukiwaniu podobnych tekstów, które mierzą znaczenie terminów w dokumentach, przy czym TF-IDF wykorzystuje częstość występowania

terminu i jego rzadkość w korpusie, a BM25 pozwala dodatkowo parametryzować funkcję saturacyjną do ważenia często występujących terminów oraz wpływ długości tekstu.

Obecnie jednak dużo bardziej rozpowszechnione od wspominanych algorytmów stało się wykorzystanie specjalizowanych modeli, wyliczających dla tekstu tzw. *embeddingi* (w języku polskim przyjęło się pojęcie "osadzenia", nie jest ono jednak precyzyjne; w tej pracy zamiennie używam pojęć *embedding* i *reprezentacja wektorowa*, zgodnie z wyjaśnieniem w dalszej części). *Embedding* dla tekstu to reprezentacja słów lub zdań jako wektorów liczbowych w przestrzeni wielowymiarowej, która umożliwia uchwycenie ich znaczenia i relacji.

Do powszechnie wykorzystywanych obecnie modeli do wyliczania reprezentacji wektorowych należą:

- Word2Vec [20].
- GloVe (Global Vectors for Word Representation) [21].
- FastText [22].
- ELMo (Embeddings from Language Models) [23].
- BERT (Bidirectional Encoder Representations from Transformers) [24].
- GPT (Generative Pre-trained Transformer) [25].
- RoBERTa (A Robustly Optimized BERT Pretraining Approach) [26].
- XLNet [27].

Reprezentacje wektorowe wylicza się zarówno dla pytania, jak i dla fragmentów [16]. Kolejnym krokiem jest sprawdzenie podobieństwa wektorów pytania i fragmentów tekstu. Do ustalenia podobieństwa używa się zazwyczaj jednej z dwóch metod. Pierwszą opcją jest wykorzystanie odległości euklidesowej pomiędzy wektorami (czym mniejsza wartość tym wektory bardziej podobne). Odległość euklidesowa dla wektorów mierzy prostą liniową odległość między dwoma wektorami w przestrzeni wielowymiarowej, obliczaną jako pierwiastek kwadratowy sumy kwadratów różnic odpowiadających sobie współrzędnych. Formalnie przedstawia się to następująco:

$$d(\mathbf{u},\mathbf{v}) = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2}$$

Druga opcja to pomiar podobieństwa cosinusowego (czym większa wartość tym wektory bardziej podobne; czasami używa się też dystansu cosinusowego). Podobieństwo cosinusowe mierzy kąt między dwoma wektorami w przestrzeni wielowymiarowej, określając ich podobieństwo jako iloraz ich iloczynu skalarnego i iloczynu ich długości, dając wartość od 0 do 1 dla wektorów o nieujemnych współrzędnych. Podobieństwo cosinusowe między dwoma wektorami $\mathbf{u}$ i $\mathbf{v}$ jest dane wzorem:

$$\cos(\theta) = \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\,\|\mathbf{v}\|}$$

gdzie:

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

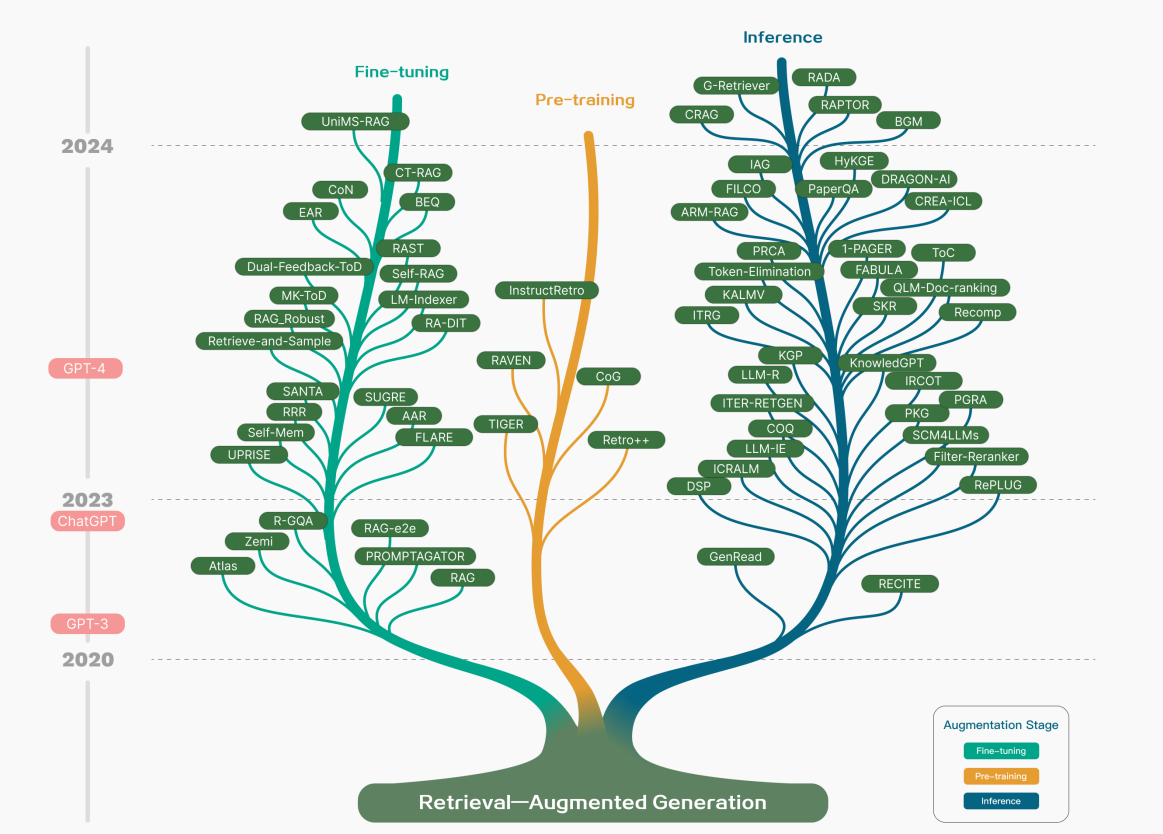
jest iloczynem skalarnym wektorów **u** i **v**, a

$$\|\mathbf{u}\| = \sqrt{\sum_{i=1}^{n} u_i^2} \quad \text{oraz} \quad \|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

to długości (normy) wektorów **u** i **v**.

Dystans euklidesowy jest przydatny do porównywania tekstów, gdy istotne są bezwzględne różnice między wektorami reprezentującymi te teksty. Podobieństwo cosinusowe lepiej sprawdza się w porównywaniu tekstów o różnych długościach, ponieważ ocenia podobieństwo kierunkowe, a nie różnice w skali. Z tego względu częściej to podobieństwo cosinusowe jest używane do porównywania kontekstów i znaczeń tekstów, szczególnie w kontekście Retrieval Augmented Generation, gdzie z reguły istnieje duża dysproporcja pomiędzy długością pytania i fragmentów.

Rozpowszechnienie RAG i intensywność prac nad ulepszeniami architektury można prześledzić na załączonej grafice:



Rysunek 2.5. Hierarchia systemów RAG (źródło: [28])

W produkcyjnie wdrażanych rozwiązaniach do obsługi porównań i poszukiwań najbliż-

szych znaczeniowo wektorów najczęściej wykorzystuje się bazy danych oferujące wsparcie dla tego typu zastosowań. Istnieją tutaj dwie możliwości. Jedna to bazy dedykowane dla przeszukań wektorowych (*vector databases*), takie jak Weaviate [29] i Pinecone [30]. Druga to rozszerzenia istniejących baz, dodające wektorowe typy danych, odpowiednie rodzaje indeksów i algorytmy do wydajnych przeszukiwań. Do tej kategorii można zaliczyć m.in. ElasticSearch [31] i PostgreSQL [32] z pluginem pgvector [33]. Dla przykładu, ten ostatni dostarcza dodatkowy typ danych *VECTOR(size)*, wbudowane funkcje do porównań dla tych danych (jak <=> dla dystansu cosinusowego), oraz indeksy używające algorytmów właściwych dla wektorów (jak HNSW, *Hierarchical Navigable Small Worlds* [34]). W bazie przechowywane są fragmenty razem w wyliczonymi wcześniej reprezentacjami wektorowymi. Kiedy zostaje wprowadzone pytanie, wyliczane są reprezentacje także dla niego, i następnie wyszukuje się określoną ilość najbliższych znaczeniowo rezultatów.

Czasami faza poszukiwania pokrewnych rezultatów wzbogacana jest o dodatkowy etap zwany *rerankingiem*, w której wyselekcjonowane wyniki podaje się dodatkowej weryfikacji i ewentualnemu przesortowaniu. Rozpowszechnione jest użycie w tym celu *cross-encoderów*. *Cross-encodery* to modele wykorzystujące głębokie sieci neuronowe, które oceniają podobieństwo lub interakcje między parami tekstów, przetwarzając oba teksty jednocześnie, co pozwala na bardziej precyzyjne dopasowanie niż metody dwustopniowe wykorzystujące osobne enkodery dla każdego tekstu. Przetwarzanie obu tekstów jednocześnie jest możliwe dzięki połączeniu ich w jedną całość i wykorzystaniu specjalnego rozdzielającego tokena. Do najbardziej popularnych modeli *cross-encoderów* należą wspomniane już BERT (*Bidirectional Encoder Representations from Transformers*) i jego wariant RoBERTa (*A Robustly Optimized BERT Pretraining Approach*). Użycie *cross-encoderów* na pełnym korpusie zazwyczaj jest niemożliwe ze względu na potrzebny w tym celu czas. Bywa jednak pomocne w finalnej ocenie i wyborze najwłaściwszych trafień spośród *top k* kandydatów wyselekcjonowanych za pomocą generalnego przeszukiwania na podstawie podobieństwa wektorów.

Opisany powyżej schemat przedstawia w zarysie najprostszą implementację koncepcji Retrieval Augumented Generation. Literatura [28] nazywa go naiwnym RAG (*naive RAG*), definiując także kolejne ewolucje: zaawansowany RAG (*advanced RAG*) i modularny RAG (*modular RAG*). Starają się one uwzględnić problemy, na które napotyka podejście naiwne, a do których należą przede wszystkim:

- **Fragmentaryczność informacji**: Wybór fragmentów na podstawie podobieństwa semantycznego do pytania pomija konieczność zrozumienia, jaką funkcję fragmenty pełnią w tekście. Prowadzi to do niespójności i niekompletności odpowiedzi.
- **Brak integracji semantycznej**: Pokrewnym problemem jest brak uchwytywania związków semantycznych pomiędzy wybranymi fragmentami. Staje się to źródłem błędów w odpowiedziach, szczególnie w wypadkach, gdy odpowiedź wymaga dobrego rozumienia zależności występujących w źródłowym tekście.

- **Ignorowanie kontekstu globalnego**: RAG bazujący tylko na fragmentach pomija kontekst globalny źródłowych tekstów, co może prowadzić do fragmentarycznej lub błędnej interpretacji wykorzystanych fragmentów.
- **Nadmierne poleganie na podobieństwie semantycznym**: Wykorzystanie podobieństwa reprezentacji wektorowych pomiędzy pytaniem i odpowiedzią w wielu wypadkach jest bardzo zawodną metodą. Nie zawsze podobieństwo pytania i fragmentu jest realną gwarancją tego, że wybrany fragment zawiera informację niezbędną z punktu widzenia generowania odpowiedzi.
- **Problemy z łączeniem sprzecznych informacji**: Wybrane fragmenty mogą zawierać sprzeczne (lub pozornie sprzeczne) informacje. Bez dodatkowych instrukcji model nie jest w stanie właściwie obsłużyć takich przypadków.

Wspomniane już systemy zaawansowanego i modularnego RAG próbują odpowiedzieć na te wyzwania w różny sposób, przede wszystkim za pomocą ulepszania podziału dokumentów, usprawnienia ich selekcji i filtrowania, a także dzięki trenowaniu dodatkowych modeli wspierających konkretne wyszukiwania. Istotnym obszarem są również próby precyzyjniejszego uwzględnienia generalnej struktury tekstu.

Ciekawym przykładem złożonego systemu RAG jest system zaprezentowany w artykule "Bailicai: A Domain-Optimized Retrieval-Augmented Generation Framework for Medical Applications"[35]. Przypadek ten skupiony jest na dostosowaniu rozwiązania dla potrzeb medycyny, dlatego dokładność i precyzja odpowiedzi ma bardzo duże znaczenie. Usprawnienia zaproponowane przez system zaczynają się już na poziomie modeli wykorzystywanych dla obliczenia reprezentacji wektorowych. Są to modele specjalnie wytrenowane na danych medycznych, tak, żeby możliwie dokładnie potrafiły budować reprezentacje wektorowe dla tej domeny. Zabrana wiedza poddana jest szczegółowej indeksacji, opartej o tablice mieszające, indeksy wektorowe i drzewa wyszukiwania. Usprawnienia występują również na poziomie samych wyszukiwań, gdyż oprócz typowego podobieństwa pomiędzy wektorami, wykorzystywane są także algorytmy K-Nearest Neighbours oraz techniki modelowania domenowego takie jak Latent Dirichlet Allocation. Zadane pytania także ulegają dodatkowemu rozszerzeniu dzięki wprowadzonym technikom wzbogacania, rozwijającym pytanie o dodatkową terminologię i powiązania. Finalnie, model językowy dostosowany jest do obszaru medycyny, czego dokonano poprzez dotrenowanie na specyficznym dla dziedziny zbiorze danych. Wszystkie te zabiegi pozwoliły skonstruować system o dużej niezawodności, a dostarczone wyniki wskazują, że może on być rzeczywiście wykorzystywany nawet w tak złożonej dziedzinie, jaką jest medycyna.

Innym przykładem jest zastosowanie RAG dla celów z zakresu prawa. Najbardziej zaawansowanym (na podstawie wyników osiąganych na zbiorze danych CAIL2018 [36]) jest rozwiązanie zaproponowane w "Athena: Retrieval-augmented Legal Judgment Prediction with Large Language Models"[37]. W tym wypadku dużą wagę przykłada się do odpowiedniego przygotowania danych wejściowych. W pierwszym kroku są one z wykorzystaniem

możliwości modeli językowych ujednolicane i dostosowywane dla potrzeb systemu. Wydzielane są też kluczowe segmenty każdego przypadku, takie jak fakty sprawy, orzeczenie i argumentacja prawna. Następnie, dokonywana jest identyfikacja nazw własnych (*NER - Named Entity Recognition*), a używane terminy są standaryzowane, tak, żeby łatwo łączyć pokrewne przypadki. Innym uprawnieniem w stosunku do podstawowego RAG są techniki pozwalające dynamicznie doprecyzować kontekst, poprzez usuwanie jego niepotrzebnych części (dokonywane za pomocą cząstkowej oceny podobieństw semantycznych oraz oceny za pomocą wykorzystywanego dużego modelu językowego). Ważnym etapem jest ostatni krok, polegający na weryfikacji uzyskanych odpowiedzi. System dokonuje dodatkowego porównania z istniejącymi przypadkami, żeby potwierdzić trafność generowanej interpretacji. Co więcej, do modelu kierowane jest polecenie oceny stopnia przekonania co do poprawności przygotowanej odpowiedzi. Pozwala to na interwencję człowieka w wypadkach, gdy istnieje niepewność i większe ryzyko błędu.

Jeszcze inne interesujące usprawnienia zaproponowano w artykule Customized Retrieval Augmented Generation and Benchmarking for EDA Tool Documentation QA [38]. Zajmuje się on usprawnieniem RAG dla celów dotyczących zadawania pytań do dokumentacji narzędzi do automatyzacji projektowania układów elektronicznych (*EDA - Electronic Design Automation*). W tym wypadku również ulepszono proces tworzenia reprezentacji wektorowych, lecz tym razem do ulepszenia modeli wykorzystano uczenie kontrastowe. Podejście to poprawiło rozumienie terminów związanych z EDA, co przekłada się na znaczną poprawę rezultatów wyszukiwania. Na znalezionych w pierwszym etapie wynikach stosowany jest dodatkowo reranking, przy użyciu modelu wydestylowanego z bazowego dużego modelu językowego, dostosowanego do selekcji najbardziej istotnych dokumentów. Również sam model, który generuje finalne odpowiedzi, został dotrenowany na dokumentach pochodzących z dziedziny EDA, a następnie dodatkowo wzmocniony dzięki dostrajaniu instruktażowemu (z wykorzystaniem wysokiej jakości par pytań i odpowiedzi specyficznych dla narzędzi EDA).

Spośród publikacji proponujących dalsze drogi rozwoju RAG bardzo ciekawy wydaje się artykuł "Blended RAG: Improving Retrieval-Augmented Generation with Semantic Search and Hybrid Query-Based Retrievers"[39]. W tym wypadku istotne są dwa komponenty. Pierwszym jest integracja wyszukiwania semantycznego, polegająca na rozszerzeniu tradycyjnego wyszukiwania opartego o gęste wektory o wyszukiwania również na wektorach rzadkich. Pozwala to lepiej uchwycić specyfikę tekstu, m.in. rzadko spotykane słowa kluczowe. Drugim komponentem jest dynamiczne dostosowanie strategii wykorzystywanej do wyszukiwania w zależności od zapytania. Pozwala to łączyć opisane rozszerzone wyszukiwanie na wektorach z metodami takimi jak BM25 w zależności od przewidywania, jaka metoda sprawdzi się w konkretnym przypadku najlepiej.

Nieco podobny system wykorzystany jest w publikacji "PersonaRAG: Enhancing Retrieval - Augmented Generation Systems with User-Centric Agents"[40]. W tym wypadku

budowany jest system agentów, potrafiących dynamicznie dostosować cały proces wyszukiwania i konstruowania kontekstu w zależności od interakcji z użytkownikiem. Agenci monitorują i przewidują wzorce zachowań oraz preferencje użytkownika. Pozwala to na lepszą personalizację i indywidualne dostosowanie wyników.

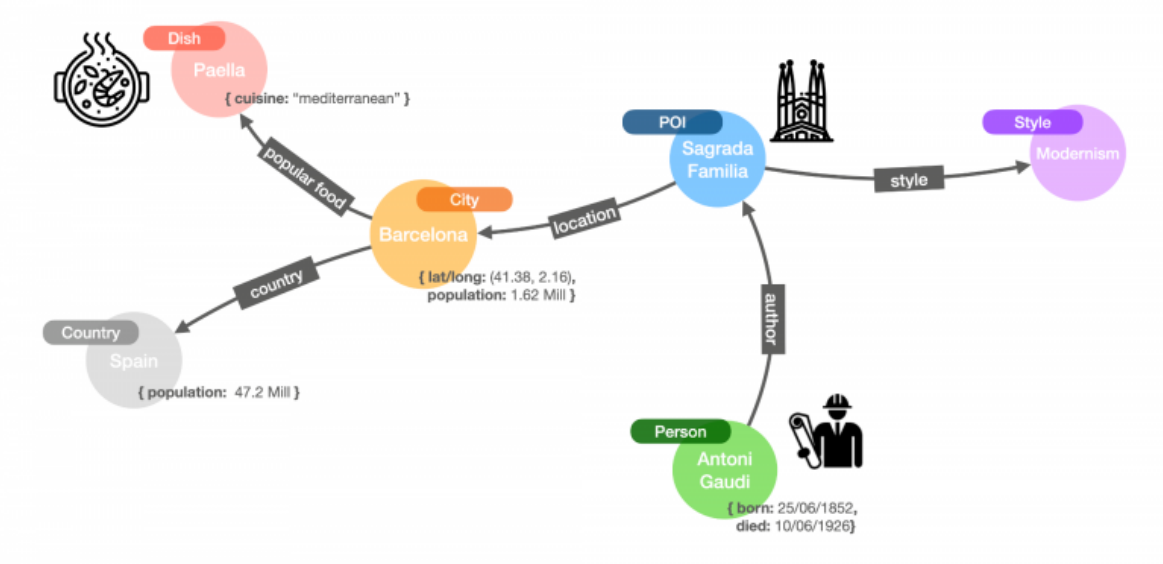
2.3. Wykorzystanie grafów w Retrieval Augmented Generation

Jednym ze ważnych obszarów poszukiwania rozwiązań, które poprawiałyby jakość wyszukiwania kontekstu, a tym samym także finalnej odpowiedzi otrzymywanej z modelu LLM, stało się wykorzystanie grafów. Poszukiwania te prowadzone są w dwóch głównych kierunkach: budowy grafów wiedzy na podstawie encji wyodrębnionych z tekstu oraz budowy grafów reprezentujących strukturę dokumentów (najczęściej wielu).

2.3.1. Grafy wiedzy

Pierwsze podejście polega na ekstrakcji ze źródłowych dokumentów grafów wiedzy (*Knowledge Graphs*). Grafy wiedzy to struktury umożliwiające formalne uchwycenie obiektów albo pojęć i relacji pomiędzy nimi. Obiekty w tym ujęciu stają się węzłami grafu, a relacje pomiędzy nimi reprezentowane są za pomocą krawędzi. Każdy obiekt i relacja może być dookreślona dodatkowymi atrybutami. Do podstawowych zalet grafów wiedzy należy precyzyjne ujęcie zależności w danej dziedzinie, ale także bardzo szerokie możliwości analizy. Współczesna informatyka opracowała dużą ilość algorytmów pozwalających skutecznie pracować z grafami wiedzy, od algorytmów pozwalających na detekcję klik i zagęszczeń w grafie, poprzez analizy lokalne i przewidywanie potencjalnych połączeń, aż po detekcję połączeń i powiązań pomiędzy odległymi nawet węzłami.

Przykładowy graf wiedzy wygląda następująco:



Rysunek 2.6. Przykładowy graf wiedzy (źródło: [41])

Od strony formalnej graf wiedzy G można zdefiniować jako graf skierowany, który składa się z zestawu encji (węzłów) oraz relacji (krawędzi) między tymi encjami. Każda relacja w grafie wiedzy jest wyrażana w postaci trójki uporządkowanej (według schematu *RDF - Resource Definition Framework*):

$$(s, p, o) \quad \text{gdzie} \quad s = \text{subject},\ p = \text{predicate},\ o = \text{object}$$

gdzie:

- s jest *podmiotem* (ang. subject) - encją będącą źródłem relacji,
- p jest *predykatem* (ang. predicate) - typem relacji między encjami,
- o jest *obiektem* (ang. object) - encją będącą celem relacji.

Zbiór encji E i zbiór relacji R definiują graf wiedzy:

$$G = (E, R)$$

Każda encja $e \in E$ może posiadać dodatkowe atrybuty $A(e)$, które opisują właściwości encji, np. wiek osoby czy lokalizację obiektu. Relacje $r \in R$ są definiowane przez trójki (s, p, o), gdzie $s \in E$, $p \in P$ (zestaw predykatów), a $o \in E$ lub $o \in V$ (zestaw wartości atrybutów).

Graf wiedzy można więc opisać formalnie jako:

$$G = \{(s, p, o) | s \in E, p \in P, o \in (E \cup V)\}$$

Ważnym pojęciem w kontekście grafów wiedzy jest pojęcie ontologii. Ontologie definiują formalne zasady dla opisu świata rzeczywistego dokonywanego przez graf. Zawierają one przede wszystkim definicję klas i pojęć wykorzystanych w grafie, obejmując również wzajemne zależności pomiędzy nimi, a także opis dostępnych atrybutów. Posługują się zazwyczaj wysoce sformalizowanym językiem, dzięki czemu możliwe jest przeprowadzanie wnioskowań i wyszukań na grafie.

Do najbardziej rozpowszechnionych ontologii należą RDF (*Resource Definition Framework*) i OWL (*Web Ontology Language*):

- RDF powstał jako model służący co opisu informacji w sieci, ale szybko został zaadaptowany do opisu grafów. Jego wyróżnikiem jest posługiwanie się zestawami trójek: podmiot, predykat i obiekt. Umożliwia to zarówno zamodelowanie relacji pomiędzy dwoma encjami, na przykład:

  $$(Platon, urodzony - w, Ateny)$$

  W tym przypadku „Platon” to podmiot, „urodzony” to predykat, a „Ateny” to obiekt. Możliwe jest także przypisanie encji określonych atrybutów:

$$(Ateny, populacja, 643452)$$

Zaletą RDF jest prostota, elastyczność i łatwość wykorzystania, a także niska komplikacja formatu danych, ułatwiająca dalsze ich przetwarzanie. Z założenia RDF korzysta z unikalnych identyfikatorów URI (Uniform Resource Identifier), co po pozwala na zachowanie globalnej jednoznaczności.

- OWL jest rozszerzeniem RDF, które pozwala na bardziej złożone modelowanie wiedzy. W tym wypadku możliwe jest ustalenie reguł logicznych i zasad wnioskowania dotyczących danych zebranych w grafie. Ważnym rozszerzeniem jest także możliwość dziedziczenia, i tym samym budowania hierarchii klas (dla przykładu, "filozof"może być klasą ogólną, a "Platon"instancją tej klasy)

W praktycznych zastosowaniach, szczególnie w grafowych bazach danych takich jak Neo4j [42] lub NebulaGraph [43], rozpowszechnionym sposobem reprezentacji grafu jest Labeled Property Graph (LPG), co można tłumaczyć jako "etykietowany graf z właściwościami". Najistotniejszą właściwością tego rodzaju reprezentacji jest użycie etykiet (*labels*), oznaczających typ (klasę) węzła i relacji. Oprócz tego zarówno węzły, jak i relacje, mogą mieć dowolną ilość dookreślających je właściwości (*properties*). Grafy tego typu, szczególnie w rozwiązaniach bazodanowych, zoptymalizowane są pod kątem przeszukań, przede wszystkim na strukturze grafu (różnego typu przejścia, wyszukiwania najkrótszej ścieżki pomiędzy zadanymi węzłami etc), ale także po etykietach lub właściwościach (w tych zastosowaniach bliskie są klasycznym relacyjnym bazom danych).

Grafy wiedzy we współczesnych naukach informatycznych mają cały szereg zastosowań [44]. Wraz ze lawinowo rosnącą ilością informacji konieczne staje się ich wydajne strukturyzowanie, do czego w wielu przypadkach grafy są najlepszym rozwiązaniem. Tradycyjne zastosowania grafów obejmują analizy różnego typu sieci społecznościowych, a także powiązań pomiędzy użytkownikami i produktami w sklepach internetowych, albo pomiędzy użytkownikami i preferowanymi treściami w różnego typu serwisach oferujących treści audio i video. Znajomość dotychczasowych struktur pozwala przewidywać i rekomendować potencjalnie interesujące produkty lub treści. Grafy wiedzy przydatne są także w analizach sieci i ruchu sieciowego. Coraz powszechniejsze jest zastosowanie grafów wiedzy w medycynie (gdzie uporządkowanie danych medycznych pozwala na lepsze powiązanie i wyszukiwanie niezbędnych faktów dotyczących badanego przypadku), edukacji (bardzo ciekawym zastosowaniem grafów wiedzy jest wykorzystanie ich do porządkowania i przeszukiwania zbiorów danych składających się z artykułów, patentów i innych treści naukowych), wykrywaniu oszustw (dzięki grafom wiedzy możliwe jest odkrycie powiązań, które mogą wskazywać na nielegalne działania) oraz w różnych dziedzinach z zakresu zarządzania przedsiębiorstwem (grafy wiedzy mogą opisywać łańcuchy dostaw, relacje z klientami, powiązania działów w przedsiębiorstwie etc).

Zdolność grafów wiedzy do precyzyjnego uchwycenia pojawiających się encji i zależ-

ności pomiędzy nimi wydaje się szczególnie przydatna w kontekście Retrieval Augmented Generation. Obecne wysiłki koncentrują się przede wszystkim na tym, by ze źródłowego tekstu lub tekstów wyekstrahować graf wiedzy, a następnie na nim przeprowadzać wyszukania kontekstów potrzebnych dla LLM. W założeniu pozwala to dostarczyć lepszych jakościowo danych niż tylko mechanicznie wydzielone fragmenty tekstu. Przykłady takich zastosowań pojawiły się w wielu dziedzinach, od predykcji zachowań uczestników ruchu drogowego, poprzez medycynę, aż po systemy obsługi klienta ([45], [46], [47], [48]). Typową metodą budowy grafu jest użycie LLM do wyodrębnienia z tekstu istotnych znaczeniowo elementów. W takim wypadku model, przy użyciu odpowiednich technik zapytania (*prompt engineering*), instruowany jest, by ze źródłowego tekstu wyodrębnić encje oraz powiązania pomiędzy nimi [49]. Uzyskane rezultaty mogą być ładowane do pamięci i przeszukiwane w trakcie działania aplikacji, ale częściej zapisywane są w dedykowanych systemach lub bazach danych, które wspierają szybkie przeszukiwania na grafach wiedzy, także z uwzględnieniem istotnych sąsiedztw. Od tego momentu przygotowany graf może stać się albo samodzielną bazą wyszukań, albo źródłem wspierającym, rozszerzającym wyniki uzyskane za pomocą opisanych wcześniej metod, takich jak BM25 lub wyszukania z użyciem wektorów reprezentacji.

Przykładem zastosowania grafów wiedzy może być propozycja przedstawiona w artykule "Cross-Data Knowledge Graph Construction for LLM-enabled Educational Question - Answering System: A Case Study at HCMUT"[46]. W tym wypadku źródłem danych do stworzenia grafu stają się dane z dokumentów będących w obiegu akademickim. Graf budowany jest w użyciem danych edukacyjnych z różnorodnych systemów, w tym z portali edukacyjnych i systemów zarządzania nauką (LMS). Artykuł wyróżnia rozbudowana metoda ekstrakcji encji i relacji a danych źródłowych, gdyż oprócz standardowych technik inżynierii zapytań jest ona rozbudowana o techniki klasteryzacji semantycznej oraz odkrywania relacji pomiędzy encjami za pomocą reprezentacji wektorowych. Finalnym celem postawionym przed grafem i korzystającym z niego modelem jest przede wszystkim stworzenie zintegrowanego środowiska edukacyjnego, ale także dostarczenie wyczerpujących danych dla wirtualnego asystenta, który potrafi udzielać odpowiedzi na pytania z uwzględnieniem kontekstu i historii osoby pytającej. Oprócz kontekstu akademickiego metoda została przetestowana również na danych z sektora bankowego, w obu wypadkach przynosząc dobre rezultaty.

Inny przykład przedstawiony jest w pracy "RAG-based Explainable Prediction of Road Users Behaviors for Automated Driving"[47]. W tym wypadku grafy wiedzy i RAG wykorzystane są dla przewidywania i interpretacji zachowań uczestników ruchu drogowego. W fazie tworzenia grafu wyodrębniane są encje (takie jak piesi, pojazdy w ruchu) i budowane relacje pomiędzy nimi. Uzyskiwane jest to za pomocą modeli analizujących obraz oraz dane tekstowe. Duży nacisk jest kładziony na pozycję i wzajemne położenie poszczególnych elementów w przestrzeni, z czego można wnioskować ich dalszy ruch. Prognozy

położenia w czasie i podejmowanych aktywności również identyfikowane są za pomocą dedykowanych modeli. Wszystkie te dane trafiają do grafu, opisującego wszystkich uczestników ruchu drogowego, ich położenie i przewidywania co do kontynuacji obecnego ruchu. Graf staje się źródłem wyszukań, a model jest w stanie nie tylko przewidywać kolejne mogące się zdarzyć sytuacje, ale także poddać je interpretacji i uzasadnić swoje przewidywania, a finalnie generować wartościowe wskazówki dla systemu autonomicznego kierowania pojazdem.

Ciekawe rozwiązanie zaprezentowane jest w artykule "HyKGE: A Hypothesis Knowledge Graph Enhanced Framework for Accurate and Reliable Medical LLMs Responses", wspierającego konstruowanie hipotez medycznych [50]. Graf wiedzy budowany jest ze źródeł o wysokiej wiarygodności, jak publikacje naukowe oraz bazy danych kliniczne i farmaceutyczne. Dodatkowo używane są zasoby terminologiczne i ontologiczne takie jak UMLS (*Unified Medical Language System*), SNOMED CT (*Systematized Nomenclature of Medicine - Clinical Terms*) oraz MeSH (*Medical Subject Headings*). Ze źródeł za pomocą ontologii ekstrahowane są węzły, które reprezentują jednostki medyczne, takie jak choroby, leki, procedury, symptomy i geny, oraz relacje pomiędzy nimi. Najważniejszym elementem rozwiązania, które odróżnia go od innych przedstawionych systemów, jest zaprojektowany moduł rerankingu. Wszystkie informacje wyciągnięte z grafu analizowane są pod kątem trafności i różnorodności. W tym celu wykorzystane są algorytmy rankingowe oraz mechanizm uwagi, z wykorzystaniem celowo trenowanego modelu. Dzięki ocenie trafności w rozwiązaniu możliwe jest odfiltrowanie nietrafnego kontekstu, ale także dostosowanie go zapytania, a dzięki ocenie różnorodności możliwe jest dostarczenie kontekstów ujmujących wiele wymiarów, co wpływa na jakość i precyzję finalnych odpowiedzi.

Bardziej rozbudowany system tej kategorii zaprezentowany został w pracy "GRAG: Graph Retrieval-Augmented Generation"[51]. Projekt ten wychodzi od obserwacji, że w istniejących rozwiązaniach niewystarczającą wagę przykłada się do topologii budowanych grafów. W procesie proponowanym przez GRAG pod uwagę brane są nie tylko wektory reprezentacji powiązane z poszczególnymi węzłami, ale dodatkowo wyodrębniane są podgrafy skupione wokół określonych istotnych encji, i także one stają się źródłem reprezentacji wektorowych, które porównywane są z zapytaniem.

Najnowszą generacją tej kategorii systemów jest projekt GraphRag ([52], [53]). W tym wypadku nie tylko konstruowany jest graf wiedzy, ale - analogicznie do projektu GRAG - encje łączone są w większe znaczeniowe klastry, w celu lepszego zachowania semantyki źródłowego tekstu. Pierwszy krok jest wspólny z pozostałymi metodami i polega na wyodrębnieniu encji i relacji, w głównej mierze z wykorzystaniem dużych modeli językowych i inżynierii zapytań. Stworzony graf jest następnie poddany dalszemu procesowaniu, a wyodrębnione encje grupowane są w większe całości i partycje (z wykorzystaniem algorytmu Leiden). Pozwala to na utworzenie wielopoziomowej struktury powiązanych

informacji. Rozwiązanie to szczególnie dobrze sprawdza się dla tekstów, gdzie występuje duże nagromadzenie faktów.

2.3.2. Hierarchiczne struktury dokumentów źródłowych

Drugą kategorią systemów wykorzystujących grafy w celu poprawienia wyników RAG są systemy bazujące na hierarchicznym strukturyzowaniu dokumentów źródłowych i występujących w nich tematów. Rozwiązania z tej kategorii mogą mieć bardzo różną postać, ale podstawową cechą jest to, że nie są zorientowane na uchwycenie encji występujących w tekstach, ale same teksty lub ich fragmenty stają się węzłami budowanego grafu. Połączenia pomiędzy węzłami budowane mogą być na różnej podstawie, od wspólnego pochodzenia, poruszanej tematyki, aż po podobieństwo semantyczne. Wykorzystanie grafów tego rodzaju dobrze sprawdza się szczególnie w sytuacjach, gdy dane źródłowe zawierają bardzo dużo dokumentów, a zadawane pytania wymagają łączenia wiedzy pochodzącej z wielu spośród nich.

Innym przykładem tej kategorii jest artykuł "Knowledge Graph Prompting for Multi - Document Question Answering"[54]. W tym wypadku węzłami stały się fragmenty dokumentów. Następnie używa się różnorodnych technik, od podobieństwa semantycznego po TF-IDF, żeby ustalić zachodzące pomiędzy nimi relacje. Stworzony graf może być trawersowany, także z wykorzystaniem możliwości dużego modelu językowego, żeby wyszukać takie ścieżki w grafie, które pozwalają na wygenerowanie najtrafniejszej odpowiedzi. Przeprowadzone eksperymenty pokazały, że generowane z wykorzystaniem takiego grafu odpowiedzi posiadają wysoką jakość, szczególnie w zastosowaniach na dużych bazach dokumentów i w wypadkach, gdy istotne są złożone zależności pomiędzy wieloma dokumentami, jak na przykład w prawie albo w finansach.

Nieco odmienne rozwiązanie tej kategorii zaprezentowane jest w artykule "Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering"[55]. W tym wypadku domeną jest obsługa klienta, a celem poprawa jej efektywności. Dane źródłowe stanowią przede wszystkim dotychczasowe zgłoszenia. Są one poddawane dwutorowej analizie, w której wykrywane są zarówno wewnętrzne zależności w dokumentach, a także relacje pomiędzy nimi, które pozwalają na utworzenie grafu. W rozwiązaniu tym wewnętrzna struktura dokumentu reprezentowana jest jako drzewo, a następnie drzewa te łączone są na podstawie podobieństw semantycznych w finalny graf. Wyszukiwanie odbywa się na podstawie reprezentacji wektorowych i kluczowych terminów znajdujących się w dokumentach źródłowych. Eksperymenty, przeprowadzone na rzeczywistych przypadkach z obsługi klienta, pozwoliły potwierdzić skuteczność rozwiązania, osiągającego rezultaty nawet o 28,6% lepsze niż dotychczasowe podejścia.

Ciekawe podejście zaprezentowane jest w pracy "Medical Graph RAG: Towards Safe Medical Large Language Model via Graph Retrieval-Augmented Generation"[56]. W tym wypadku grafy łączące wiele dokumentów wykorzystane są w medycynie. Konstruowany graf składa się z dokumentów różnorodnych typów. Najniższy poziom stanowi osobista

dokumentacja medyczna użytkowników. Na kolejnym poziomie znajdują się dokumenty zawierające ogólnodostępną wiedzę z różnych możliwie wiarygodnych źródeł, takich jak podręczniki akademickie i artykuły naukowe, przede wszystkim z uznanych czasopism i konferencji. Uzupełnienie stanowią dokumenty związane z ogólnoprzyjętymi systemami terminologicznymi, takimi jak UMLS (*Unified Medical Language System*), które zawierają definicje terminów medycznych i zależności pomiędzy nimi. Dzięki budowie grafu model jest w stanie trafnie i z niewielkim ryzykiem błędów diagnozować przypadki pacjentów, korzystając z ugruntowanej wiedzy medycznej z wielu źródeł.

Jak widać z powyższych przykładów, rozwiązania z tej kategorii cechuje bardzo duże zróżnicowanie. Zaletą większości zaprezentowanych przykładów jest stosunkowo łatwa rozszerzalność, co pozwala na wykorzystanie proponowanych podejść w różnorodnych zastosowaniach, co dodatkowo rozszerza gamę możliwości oferowanych przez duże modele językowe.

2.3.3. RAPTOR

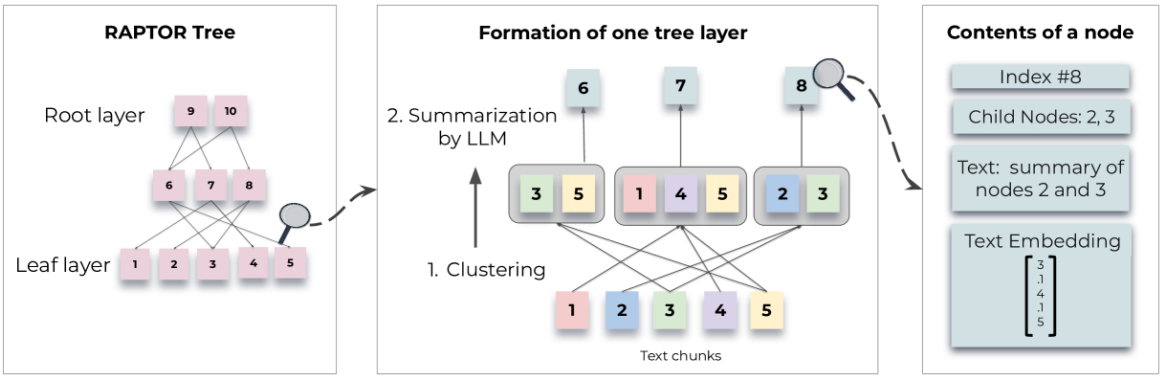
Osobne podejście do tematu zostało przedstawione w projekcie RAPTOR [57]. W tym wypadku nie skorzystano z metody polegającej na ekstrakcji encji i relacji pomiędzy nimi. Nie jest też budowany pełen graf, ale drzewo. W tym wypadku węzłami drzewa stają się fragmenty tekstu oraz streszczenia klastrów zbierających fragmenty o podobnym znaczeniu. Pozwala to na odwzorowanie złożoności tekstu na wielu poziomach ogólności, oraz na lepsze uchwycenie występujących w tekście zależności.

Pierwszy etap wspólny jest z naiwnym RAG. Źródłowy dokument dzielony jest na fragmenty zadanej długości, i dla każdego z nich wyliczana jest reprezentacja wektorowa. Fragmenty te stają się węzłami najniższego poziomu w budowanym drzewie.

Kolejnym etapem jest klastrowanie fragmentów o wysokim podobieństwie semantycznym (co wynika z założenia, że fragmenty takie uchwytują podobne idee i koncepty). Następnie, utworzone klastry poddawane są sumaryzacji z pomocą wybranego dużego modelu językowego (w źródłowym artykule głównie GPT-3.5). Wykorzystywane są modele ogólnego zastosowania, a sumaryzacja uzyskiwana jest za pomocą inżynierii zapytań. Następnie, dla każdego podsumowania obliczana jest reprezentacja wektorowa. Stworzone w ten sposób podsumowania stają się węzłami następnego poziomu.

Proces powtarzany jest iteracyjnie na kolejnych warstwach, tj. w kolejnym kroku do klastrów wybierane są sumaryzacje z poprzedniego etapu. Proces powtarzany jest dopóki istnieje wystarczająca ilość węzłów do storzenia następnego poziomu, lub do osiągnięcia zadanej wcześniej ilości warstw.

Schemat budowy drzewa jest przedstawiony na poniższym rysunku:

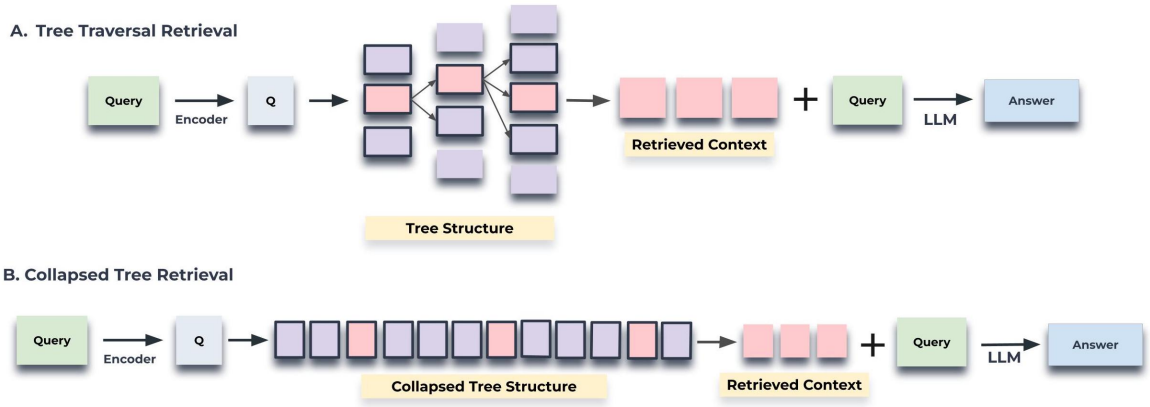


Rysunek 2.7. Schemat budowy grafu (za [57])

W projekcie RAPTOR sprawdzono dwa podejścia do wyszukiwania informacji w stworzonym drzewie. Są to:

- **Trawersowanie drzewa (*Tree Traversal*)**: W tym wypadku w pierwszej kolejności wyszukiwane są węzły z najwyższego stopnia ogólności, wybierany jest ten o największym podobieństwie semantycznym do pytania. Następnie, wyboru dokonuje się na niższym poziomie ogólności, ale wyłącznie spośród węzłów związanych z wybranym węzłem wyższego poziomu. Procesu dokonuje się aż do osiągnięcia najniższego poziomu, tj. fragmentów źródłowych. Wszystkie węzły z takiego przejścia traktowane są jako kontekst.
- **Metoda powalonego drzewa (*Collapsed Tree*)**: W tym podejściu węzły wszystkich poziomów traktowane są jednakowo, czyli w praktyce podczas wyszukiwania struktura drzewiasta jest pomijana. Wybór dokonywany jest na zasadzie podobieństwa semantycznego spośród wszystkich węzłów, niezależnie, jakie miejsce zajmowały one w strukturze. Dzięki temu możliwe jest lepsze dostosowanie wybranych węzłów do stopnia ogólności zadanego pytania. Ilość wybieranych dopasowań sterowana jest parametrem.

Obie metody można prześledzić na poniższym schemacie:



Rysunek 2.8. Schemat wyszukiwań z użyciem trawersowania drzewa (*Tree Traversal*) i metody powalonego drzewa (*Collapsed Tree*) (za [57])

W trakcie badań udowodniono, że większą skuteczność ma wykorzystanie metody powalonego drzewa.

Autorzy rozwiązania przeprowadzili eksperymenty, które jasno wskazały, że sprawdza się ono bardzo dobrze, szczególnie w wypadkach, gdy operuje się na długich i bardzo długich tekstach, a zadawane pytania nie dotyczą tylko pojedynczych faktów, ale wymagają zrozumienia całości tekstu, oraz pojawiających się w nim zależności i powiązań. Dla niektórych datasetów zbudowanych w opisany sposób (takich jak QuALITY [58]) rozwiązanie jest obowiązującym stanem rozwoju (*state-of-the-art*). To właśnie RAPTOR stał się podstawą dla przygotowanych projektu i pracy.

3. Opis eksperymentu

3.1. Cel i zakres pracy

Opisany powyżej obecny stan badań wskazuje, że wciąż pozostaje duży margines dla usprawnień dotyczących generowania odpowiedzi z użyciem RAG. Niniejszy projekt stanowi propozycję poprawy jakości wyszukań kontekstów dla tekstów narracyjnych dzięki rozwinięciu i ulepszeniu koncepcji zaprezentowanych w przedstawionych powyżej pracach, w szczególności tych związanych z grafową strukturyzacją wiedzy. Zaproponowana metoda wykorzystuje ideę projektu RAPTOR, by skorzystać z sumaryzacji dla uchwycenia relacji pomiędzy bazowymi fragmentami tekstu. Kluczową jest obserwacja, że jednoaspektowa sumaryzacja nie zawsze jest wystarczająca, szczególnie dla skomplikowanych, wielopoziomowych tekstów. Z tego względu bieżąca praca proponuje udoskonaloną strukturę grafową, dzięki której możliwe jest ukierunkowanie uwagi na różne aspekty tekstu oraz uchwycenie zróżnicowanych relacji.

Zakres i wkład badawczy projektu:

- opracowanie koncepcji grafu poprawiającego jakość wyszukań kontekstów dla tekstów narracyjnych oraz implementacja narzędzi pozwalających na jego budowę i użycie;
- zastosowanie metod inżynierii zapytań dla dobrania najlepszych zapytań dla sumaryzacji podczas konstrukcji grafu, pozwalających na optymalne działanie rozwiązania;
- porównanie wpływu parametrów wejściowych na jakość wyszukiwania i jego efektywność kosztową;
- adaptacja istniejących datasetów (QuALITY [58], NarrativeQA [59]) dla potrzeb odpowiedzi na pytania o charakterze otwartym oraz implementacja benchmarków;

3.2. Ogólna koncepcja usprawnień

Zasadnicze znaczenie dla opracowanego projektu ma charakter tekstów narracyjnych. Tekst narracyjny można zdefiniować jako rodzaj tekstu opowiadający o wydarzeniach lub sytuacjach, zazwyczaj w sposób chronologiczny. Głównym celem tego typu tekstu jest przedstawienie historii. Historia ta może być fikcyjna lub oparta na rzeczywistych zdarzeniach. Narracja prowadzona jest przez narratora, który może być uczestnikiem wydarzeń, zewnętrznym obserwatorem albo wszechwiedzącym opowiadaczem [60].

Teksty tego typu są źródłem dużych wyzwań i problemów dla dużych modeli językowych i architektury Retrieval Augmented Generation. W pierwszej kolejności są to problemy wspólne z innymi długimi tekstami, w tekstach narracyjnych występujące jednak w większym natężeniu. Do tej klasy należy zaliczyć przede wszystkim potencjalną nieciągłość narracji (najbardziej wyraźną, gdy różne wątki przeplatają się pomiędzy rozdziałami, a

nawet w ich obrębie), różnego typu przeskoki czasowe, retrospekcje, analepsy, zapowiedzi, proleksy oraz nieciągłości, pozostawiające pole dla domysłów dla czytelnika.

Na tym problemy jednak się nie kończą. W wielu utworach narracyjnych duże znaczenie ma również charakter bohaterów, zrozumienie ich motywacji, sposobów myślenia i działania. Kolejną warstwą komplikacji jest związane z tym zróżnicowanie punktów widzenia; bywa, że narracja prowadzona jest z perspektywy różnych bohaterów, co dodatkowo wpływa na rozumienie wydarzeń. Co więcej, zarówno charakter jak i punkt widzenia może zmieniać się w trakcie opowieści.

Następnym aspektem jest tło opowieści. Można tutaj wyróżnić zarówno bezpośrednie otoczenie wydarzeń, które może mieć istotny wpływ na przebieg opowieści, ale także szersze tło akcji. To ostanie może odpowiadać pewnym realiom i zależnościom historycznym, ale także może być swobodną kreacją autora (chociaż w wielu wypadkach silnie inspirowaną określonymi realiami ze świata rzeczywistego). W wielu wypadkach oba podejścia się łączą i w kontekście realiów historycznych umieszczone jest uniwersum wykreowane przez wyobraźnię autora.

Inne problemy z charakterystyką tekstów narracyjnych wiążą się z używanym językiem i stylem opowieści. Łączy się z tym zastosowanie różnego typu figur retorycznych, takich jak metafory, hiperbole, personifikacje, a także używanie symboli, alegorii i ironii. W wielu wypadkach właśnie sposób posługiwania się językiem pozwala na zrozumienie ukrytego sensu narracji.

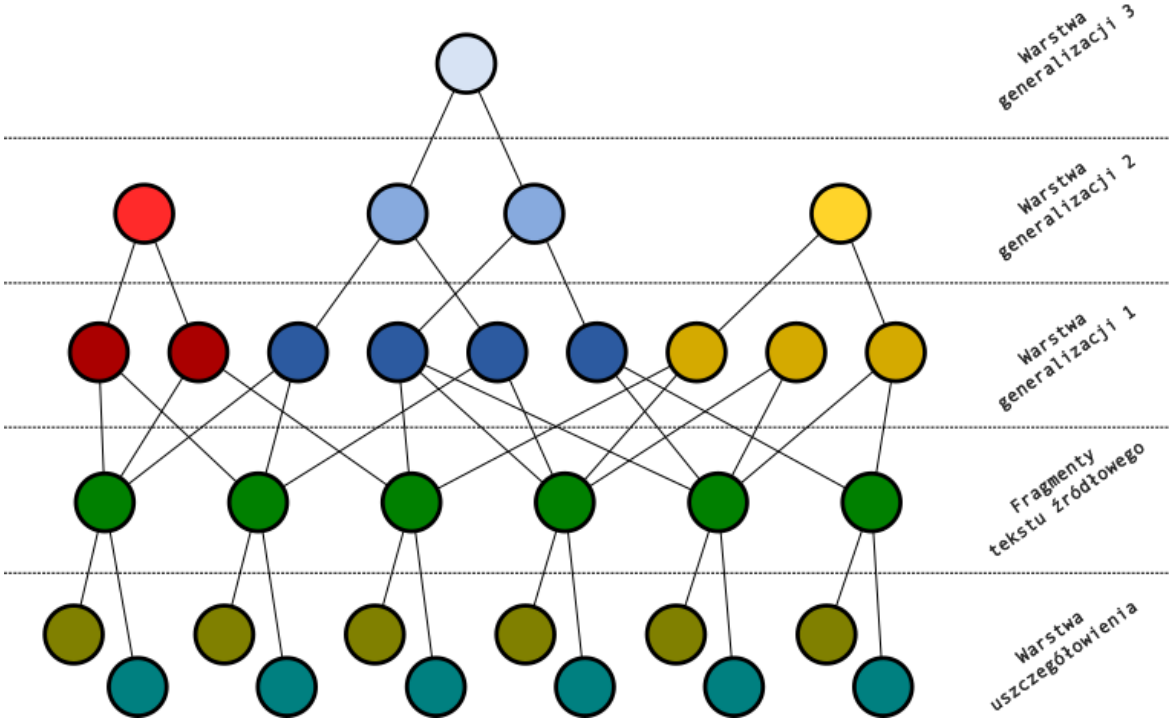
W zgodzie z tymi obserwacjami, DiYanni wyróżnia następujące kluczowe elementy tekstów narracyjnych [60]:

- Fabuła i struktura (Plot and Structure)
- Postać (Character)
- Otoczenie (Setting)
- Punkt widzenia (Point of View)
- Język i styl (Language and Style)
- Tło (Theme)
- Ironia i symbol (Irony and Symbol)

Każdy z wymienionych elementów jest aspektem spojrzenia na tekst i wariantem jego rozumienia. Identyfikacja tych elementów daje fundament do opracowania narzędzia pozwalającego skuteczniej wykorzystać możliwości Retrieval Augumented Generation dla tekstów narracyjnych. Podejście to zakłada, że spojrzenie na tekst tylko w jednym aspekcie nie pozwala uchwycić całości znaczeń i przez to obarczone jest dużym błędem. Zamiast tego, wprowadzono możliwość analizy z uwzględnieniem wszystkich opisanych aspektów, co znacząco poprawia zdolność modelu językowego do poprawnej odpowiedzi nawet na trudne pytania, wymagające zrozumienia całości tekstu, łącznie z ideami nie wyrażonymi wprost.

Wyodrębnione elementy umożliwiają budowę grafu sumaryzacji zorientowanych na

określone aspekty analizowanego tekstu. Jego podstawą jest wielopoziomowa struktura drzewiasta, zgodna z metodologią wprowadzoną w projekcie RAPTOR. Jej warstwą początkową są fragmenty tekstu. Fragmenty te są następnie grupowane są w klastry na podstawie bliskości znaczeniowej, z wykorzystaniem reprezentacji wektorowych. W celu ujęcia tekstu z wielu perspektyw przygotowano odrębne instrukcje podsumowania, osobno dla każdego z aspektów. Każde zapytanie stanowi osobną instrukcję, kładącą nacisk na charakterystyczne cechy procesowanego aspektu. Uzyskana odpowiedź staje się węzłem na wyższym poziomie. Proces prowadzony jest iteracyjne, aż do uzyskania bardzo wysokiego poziomu ogólności. Ze względu na wprowadzenie wielu aspektów, budowana struktura jest złożona z wielu drzew (każdy aspekt prowadzi do zbudowania odrębnego drzewa) połączonych przez warstwę fragmentów i przyjmuje postać grafu.



Rysunek 3.1. Schemat podziału informacji

Zbudowany graf i przygotowane podsumowania są bezpośrednią bazą wyszukań dla prezentowanego rozwiązania Retrieval Augmented Generation. Optymalnym wariantem jest przeszukiwanie w trybie zbliżonym do położonego drzewa (*collapsed tree*), tj. przeszukiwanie z użyciem podobieństw reprezentacji wektorowych na wszystkich węzłach jednocześnie, niezależnie od ich poziomu w całości struktury.

W toku badań okazało się, że nie tylko podsumowania pomagają w poprawie jakości uzyskiwanych rezultatów. Dla niektórych tekstów korzystne jest także dołożenie do grafu dodatkowych węzłów, w których umieszczane są skondensowane informacje z kolejnych fragmentów tekstu. Do kondensacji można użyć tych samych kategorii, co w przypadku podsumowań, w tym wypadku jednak model językowy pytany jest o wyciągnięcie klu-

czowych informacji i prezentację ich w możliwie zwięzłej formie (także punktów albo równoważników zdań). Dzięki temu zabiegowi część informacji jest dużo bardziej dostępna podczas fazy wyszukiwania, co przekłada się na lepsze odpowiedzi.

Zaletą prezentowanego podejścia jest nie tylko poprawa jakości odpowiedzi. Ważnym zyskiem jest także istotne zwiększenie efektywności kosztowej wyszukiwania. W wypadku omawianego rozwiązania największy koszt generowany jest podczas budowy grafu, natomiast precyzja porządkowania informacji pozwala znacznie ograniczyć ilość wyszukiwanych fragmentów, które przekazywane są do kontekstu pytania (dobre rezultaty osiąga się nawet przy podaniu jednego lub dwóch fragmentów). Aspekty te będą dokładniej przedstawione i omówione w części poświęconej rezultatom.

3.3. Narzędzia wykorzystane do implementacji

Projekt zaimplementowano przy użyciu języka Python (v3.12). Do zarządzania zależnościami wykorzystano managera Poetry (v1.8). Wykonywalny kod do przeprowadzenia poszczególnych faz procesu dostarczono za pomocą notatników Jupyter. Bazą projektu stał się projekt RAPTOR, który nie jest wersjonowany - wykorzystano wydanie, dla którego najnowszym commitem jest *2e3e8*.

Z najważniejszych użytych bibliotek należy wymienić przede wszystkim:

- ***OpenAI Python API*** *(nazwa w The Python Package Index (PyPI): openai, v1.3.3)* - do obsługi modeli udostępnionych przez OpenAI;
- ***NumPy*** *(numpy, v1.26.3)* - do złożonych obliczeń;
- ***tiktoken*** *(tiktoken, v0.5.1)* - dla potrzeb tokenizacji BPE z wykorzystaniem modeli OpenAI;
- ***tenacity*** *(tenacity, v 8.5.0)* - dla powtórnych zapytań do API i zwiększenia niezawodności aplikacji;

Preprocessing, przygotowanie danych oraz proces odpytywania przeprowadzono na dwóch modelach dostarczonych przez OpenAI: ***GPT-3.5 Turbo*** (domyślnie, model używany jeśli nie zaznaczono inaczej) oraz ***GPT-4*** (w wybranych wypadkach, dla uzyskania lepszej szczegółowości odpowiedzi). Porównanie używanych modeli można znaleźć w dołączonej tabeli.

Tabela 3.1. Porównanie wykorzystanych modeli

Model	Długość kontekstu	Dane treningowe	Ilość parametrów
GPT-3.5 Turbo	16,385 tokenów	do września 2021	20 miliardów
GPT-4	128,000 tokenów	do grudnia 2023	1.7 biliona

3.4. Szczegóły implementacji

Kluczowe zadania, które zostały zrealizowane podczas implementacji, można przedstawić w następujących punktach:

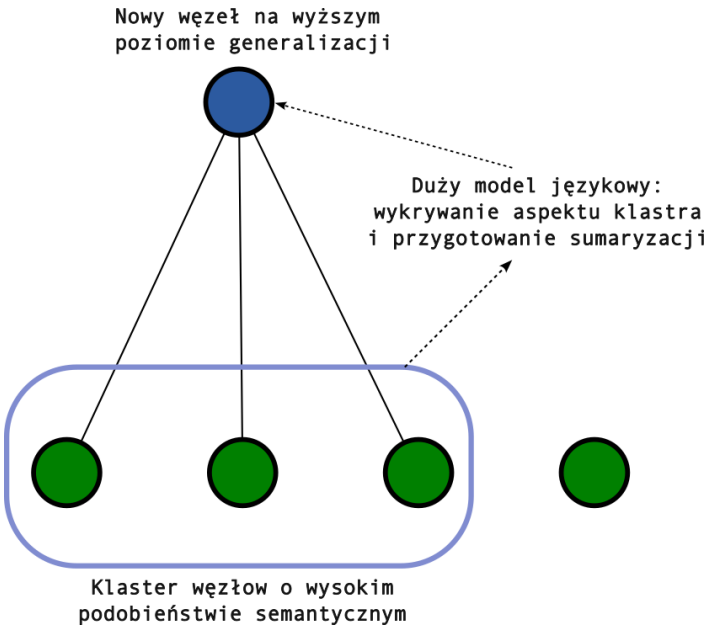
3.4.1. Metoda budowy grafu

Pierwszym krok jest wspólny z innymi rozwiązaniami RAG. Cały tekst dzielony jest na fragmenty, dla których wyliczane są wektory reprezentacji. W wypadku prezentowanego rozwiązania, fragmenty te stają się węzłami pierwszego poziomu.

Istotnym zadaniem do rozwiązania podczas budowy grafu było opracowanie systemu skutecznego wyboru fragmentów do zgrupowania w klastry, które następnie podlegają sumaryzacji. W generalnym zarysie metodę zaczerpnięto z projektu RAPTOR. Wprowadził on podejście, w którym ilość klastrów nie jest z góry ustalona, a węzły mogą należeć do wielu klastrów jednocześnie (co ma duże znaczenie, gdyż niezbędna bywa reużywalność kluczowych fragmentów). Do tworzenia klastrów algorytm wykorzystany przez RAPTOR bazuje na reprezentacjach wektorowych. W każdym klastrze znajdują się fragmenty o zbliżonej reprezentacji wektorowej. Maksymalna ilość fragmentów ograniczona jest parametrem, który z reguły koreluje z maksymalną długością kontekstu modelu (tak, by cała treść wybranych fragmentów mogła być przekazana do modelu w celu sumaryzacji).

W opracowanym rozwiązaniu następnym krokiem jest ustalenie, jakie aspekty są najwyraźniej obecne w każdym z klastrów. W tym celu również wykorzystano możliwości dużego modelu językowego (przy czym w tym wypadku ważne jest, żeby używać możliwie zaawansowanego modelu, potrafiącego dobrze wychwytywać znaczenia). Model instruowany jest o istniejących aspektach i pytany o identyfikację, które aspekty najlepiej pasują do opisanego klastra. Po tej fazie następuje dodatkowe sprawdzenie, czy każdy aspekt ma swoją reprezentację. W rzadkich (poniżej 5%) wypadkach, gdy któregoś z aspektów brakuje, cały proces jest powtarzany od nowa.

Następnie dla każdego z klastrów tworzone jest jedno lub wiele tematycznych podsumowań (zależnie od ilości wykrytych aspektów). Podsumowania te, wraz z wyliczonymi wektorami reprezentacji, stają się treścią węzłów kolejnego poziomu. Budowa klastrów powtarzana jest iteracyjnie na coraz to wyższych poziomach grafu, w obrębie ustalonych aspektów, aż do wyczerpania możliwości tworzenia sensownych klastrów lub do osiągnięcia limitu warstw, który może być zadany parametrem (domyślnie równy jest on 5).



Rysunek 3.2. Schemat tworzenia węzłów wyższego poziomu

Dodatkowo, po stworzeniu grafu podsumowań dodawane są węzły zawierające uszczegółowienia. W tym wypadku iteruje się po wszystkich węzłach stworzonych z bazowych fragmentów, i do każdego z nich dodaje określoną ilość dodatkowych węzłów.

Wyszukiwania odbywają się za pomocą podobieństwa cosinusowego pomiędzy reprezentacjami wektorowymi pytania i węzłów na wszystkich węzłach równocześnie, analogicznie do metody *collapsed tree* z projektu RAPTOR. Również w wypadku niniejszej pracy przeprowadzono eksperymenty polegające na trawersowaniu grafu od najbardziej ogólnych warstw do najbardziej szczegółowych, lub na odwrót, ale wyniki żadnego z tych podejść nie przewyższają metody bazowej.

3.4.2. Klasy odpowiedzialne za sumaryzację

Od strony kodu bardzo ważne znaczenie mają przede wszystkim komponenty odpowiedzialne za sumaryzację. Celem pracy było stworzenie modularnego systemu, by łatwiej przeprowadzić wszystkie niezbędne eksperymenty, ale także by zapewnić możliwie bezproblemową rozszerzalność w wypadku, gdyby prace nad tematem miały być kontynuowane.

W tym celu zaproponowano abstrakcyjną klasę bazową (*Abstract Base Class*), z metodą abstrakcyjną *summarize*, przyjmującą kontekst (czyli treść klastra) oraz maksymalną ilość oczekiwanych tokenów dla podsumowania (domyślnie 200):

Listing 1. Abstract Base Class (ABC) dla klas odpowiedzialnych za sumaryzację (za [57])

```
20 class BaseSummarizationModel(ABC):
21     @abstractmethod
22     def summarize(self, context, max_tokens=200):
23         pass
```

Wszystkie klasy służące do podsumowań muszą rozszerzać tę klasę bazową. Zabieg ten ułatwia budowę grafu. Instancje wszystkich implementacji są elementami listy, po której następuje iteracja w trakcie budowy grafu. Do każdej z instancji kolejno przekazywane są pasujące klastry, a wyniki po przeprocesowaniu dodawane są do grafu.

Przykładowa implementacja metody *summarize* z klasy bazowej może mieć następującą postać (zmienną *prompt* ominięto, będzie ona szczegółowo opisana w dalszej części omówienia):

Listing 2. Przykładowa uproszczona implementacja metody *summarize* (zapytanie zostało pominięte, przykładowe zapytania zostaną zaprezentowane w kolejnych listingach).

```
145 def summarize(self, context, max_tokens=200):
146
147     client = OpenAI()
148
149     prompt = ...
150
151     response = client.chat.completions.create(
152         model=self.model,
153         messages=[
154             {
155                 "role": "system",
156                 "content": "You are master in summarization of text."
157             },
158             {
159                 "role": "user",
160                 "content": prompt,
161             },
162         ],
163         max_tokens=max_tokens,
164     )
165
166     return response.choices[0].message.content
```

Do najważniejszych elementów należą inicjalizacja klasy klienta, odpowiedzialnej za obsługę zapytań do modelu (w tym przypadku *client = OpenAI()*; sam model został zainicializowany wcześniej i jest przechowany w zmiennej *model*. Następnie klient wykorzystany jest do przesłania zapytania, a zawartość odpowiedzi zwracana jest jako wynik wykonania całej metody. Zgodnie z zasadą opisaną wcześniej, wynik ten wykorzystywany jest jako zawartość kolejnego węzła w grafie.

Warto zwrócić uwagę na wykorzystanie zapytania systemowego (linie 155-156), który ma widoczny wpływ na jakość otrzymywanych odpowiedzi. Zaprezentowana powyżej nieskomplikowana forma: *"You are master in summarization of text"* sprawdziła się dobrze w większości badanych przypadków. Dobre efekty przynosi również ukierunkowanie zapytania systemowego na analizowany w klasie aspekt rozumienia tekstu. Ze względu na

modularną budowę całości systemu zmiany tego typu są stosunkowe łatwe do wprowadzania, zależnie od aktualnych potrzeb.

Najważniejszym komponentem klasy jest jednak główne zapytanie użytkownika, kierowane do modelu. Może ono przybrać następującą postać:

Listing 3. Przykładowy prompt do sumaryzacji, w tym wypadku ukierunkowanej na postacie

```
76 prompt = f"Write a summary of the following, including as many key details as
77 possible. Be very concise and strict, but include all important details. Focus
78 solely on characters and interactions between them. Here is the text to
79 summarize: {context}"
```

Przygotowanie odpowiednich zapytań było przedmiotem licznych eksperymentów, gdyż ich kształt przekłada się na jakość całego rozwiązania. Do najważniejszych obserwacji należą te, że przygotowane zapytanie powinno dobrze, lecz zwięźle definiować zadanie, precyzyjnie podkreślić najważniejsze charakterystyki określonego aspektu, oraz wprost poinstruować model, żeby skupił się właśnie na tym elemencie. Zbyt krótkie instrukcje sprawiają, że podsumowania stają się zbyt ogólne i mają ograniczoną ilość detali. Przy zapytaniach zbyt długich model nie jest w stanie uwzględnić wszystkich instrukcji, przez co zdarza się, że omija te, które mają kluczowe znaczenie.

Ważnym parametrem mającym wpływ na jakość streszczenia jest temperatura generacji. Zauważono, że przy dużych wartościach model ma skłonność do zmniejszania zgodności wygenerowanego streszczenia z danymi źródłowymi, a także zdarzają się halucynacje, polegające na dokładaniu informacji, które w rzeczywistości nie występowały w materiale źródłowym. Z tych powodów w eksperymentach posługowano się niską temperaturą (domyślnie 0.5).

Ciekawe obserwacje przyniosły próby z opisanym wcześniej *few-shot promptingiem* i próbą ukierunkowania modelu w jaki sposób powinien uchwytywać każdy aspekt. Wynik eksperymentu był zależny od oczekiwanej długości odpowiedzi. W wypadku krótkich odpowiedzi efekty były niezadowalające, gdyż model w większości wypadków potrafi wygenerować wystarczająco syntetyczne i wyczerpujące podsumowanie bez dodatkowych instrukcji, a szczegółowe przykłady raczej ograniczały tę zdolność. W wypadku długich odpowiedzi wygenerowany tekst był dobrej jakości i w bardzo wyczerpujący sposób potrafił opisać oczekiwany aspekt. Jednakże, w toku badań potwierdzono, że w praktyce używanie długich podsumowań nie jest korzystne dla finalnej jakości odpowiedzi. Z tego względu w ostatecznym wariancie ich nie zastosowano. Bardziej szczegółowo zjawisko to będzie przeanalizowane w rozdziale dotyczącym eksperymentów.

3.4.3. Klasy odpowiedzialne za uszczegółowienia

Klasy odpowiedzialne za budowę tekstów ze szczegółami zbudowane są analogicznie, jedyne zasadnicze różnice tkwią w inżynierii zapytań. Przykładowe ogóle zapytanie może wyglądać następująco:

Listing 4. Przykładowe zapytanie do przygotowania tekstu ekstrahującego najważniejsze szczegóły z danego fragmentu

```
20 prompt = f"Get details from following text: {context}. It should be as simple
21 and concise as possible, regarding both grammar and vocabulary, but it is
22 obligatory to keep all the important details. The aim is to preserve all
23 key details in simplified form. You do not have to use full sentences, just
24 point out key points."
```

Generalnie, również w tym przypadku potwierdziły się obserwacje dotyczące długości i szczegółowości instrukcji. Najlepiej sprawdziły się te średniej długości, z wprost podanymi najważniejszymi oczekiwaniami (ale bez nadmiernej ich ilości). Eksperymenty pokazały, że odpowiedzi modelu na tego typu zapytania przyjmują bardzo różną formę, od całościowej wypowiedzi, poprzez równoważniki zdań oddzielone przecinkami, aż po listę punktów. Wszystkie te formy są akceptowalne i w identyczny sposób wpływają na końcowe rezultaty.

4. Wyniki

4.1. Narzędzia i zestawy danych

Do przeprowadzenia pomiarów użyto frameworka RAGAS [61]. RAGAS (od *Relevance, Accuracy, Grammaticality, Adherence to instructions, Safety* - Trafność, Dokładność, Poprawność gramatyczna, Przestrzeganie instrukcji, Bezpieczeństwo) jest popularnym i szeroko wykorzystywanym narzędziem do badania jakości działania dużych modeli językowych i rozwiązań budowanych z ich wykorzystaniem. Oferuje całą gamę metryk, pozwalających badać rozwiązanie w każdym z wymienionych w nazwie obszarów. RAGAS jest także powszechnie wykorzystywany w rozwiązaniach komercyjnych, do budowy testów dla systemów bazujących na możliwościach dużych modeli językowych.

Spośród metryk dostępnych w frameworku RAGAS w badanym przypadku szczególnie istotne są dwie, pozwalające na ocenę poprawności wygenerowanej odpowiedzi w stosunku do podanej odpowiedzi oczekiwanej (*ground truth*, *golden answer*):

- ***Podobieństwo semantyczne odpowiedzi*** *(Answer Semantic Similarity)*. Metryka wykorzystująca do oceny reprezentacje wektorowe. Generowane są reprezentacje zarówno dla odpowiedzi wygenerowanej, jak i odpowiedzi oczekiwanej, a następnie wyliczane jest podobieństwo cosinusowe pomiędzy uzyskanymi wektorami. Czym wyższa uzyskana wartość, tym większa zgodność pomiędzy odpowiedzią wygenerowaną i oczekiwaną.
- ***Poprawność odpowiedzi*** *(Answer Correctness)*. Metryka łącząca opisane powyżej podobieństwo semantyczne z miarą F1 (*F1 score*). W pierwszym kroku obliczane jest podobieństwo semantyczne, za pomocą metody opisanej powyżej. Następnie, wyliczane jest *F1 score*, zgodnie ze wzorem:

  $$\text{F1 Score} = \frac{|\text{TP}|}{|\text{TP}| + 0.5 \times (|\text{FP}| + |\text{FN}|)}$$

  gdzie *TP* to *true positive* (informacje obecne w odpowiedzi i zgodne z odpowiedzią oczekiwaną), *FP* to *false positive* (informacje obecne w odpowiedzi, ale niezgodne z odpowiedzią oczekiwaną), *FN* to *false negative* (informacje, których w odpowiedzi brakuje, mimo, że zgodnie z odpowiedzią oczekiwaną powinny się tam znaleźć). Ocena obecności lub nieobecności informacji jest dokonywana za pomocą wybranego dużego modelu językowego. Finalna wartość metryki to średnia ważona podobieństwa semantycznego (domyślnie 25%) i miary F1 (domyślnie 75%).

Obie metryki pozwalają na uchwycenie zgodności odpowiedzi względem odpowiedzi oczekiwanej. W praktyce poprawność odpowiedzi jest metryką precyzyjniejszą, pozwalającą lepiej ocenić obecność istotnych faktów i nieobecność fałszywych stwierdzeń w odpowiedzi wygenerowanej przez model. Z tego względu w większości przeprowadzonych eksperymentów została potraktowana jako metryka bazowa.

Pomiarów dokonano na datasetach QuALITY [58] oraz NarrativeQA [59], dostępnych publicznie wraz z powiązanymi artykułami:

- ***QuALITY***. Zbiór danych służących do oceny jakości rozumienia długich tekstów różnego typu i poprawności odpowiedzi na ułożone do nich pytania. Wykorzystywany jest do ewaluacji dużych modeli językowych. Głównym celem stawianym respondentom tworzącym pytania było takie ich przygotowanie, by odpowiedź była niemożliwa bez dobrej znajomości całego tekstu, obejmującej fabułę i relacje. Dzięki takiemu ukierunkowaniu pojedyncze fragmenty tekstu zazwyczaj nie są wystarczające, by udzielić poprawnej odpowiedzi. Zbiór ma postać tekstów i pytań, do których dołączono po cztery opcje odpowiedzi.
  Dodatkową cechą QuALITY jest wyróżnienie podzbioru pytań trudnych. To pytania, dla których nawet respondenci-ludzie znający cały tekst mieli niski współczynnik poprawnych odpowiedzi w warunkach, kiedy ich czas na zastanowienie i dodatkowe przeszukiwanie tekstu był ograniczony.
- ***NarrativeQA***. NarrativeQA jest zbiorem długich tekstów zbudowanym pod kątem oceny zrozumienia długich tekstów przez modele językowe. Składają się na niego głównie całe książki i scenariusze filmowe. Pytania i odpowiedzi zostały ułożone przez respondentów, standardem jest podawanie dwóch wariantów odpowiedzi (sens pozostaje ten sam, ale sformułowanie odpowiedzi jest inne, co pozwala na większą elastyczność w ocenie). Również w tym wypadku nacisk położony został na to, by do odpowiedzi nie wystarczyła znajomość tylko lokalnego kontekstu. Zazwyczaj niezbędna jest znajomość całej fabuły, dobra orientacja w historii i faktach wyrażonych nie wprost, oraz zrozumienie relacji występujących pomiędzy postaciami, a także pomiędzy postaciami a miejscami i obiektami.

Datasety zostały zmodyfikowane dla potrzeb eksperymentu. Z obu zbiorów dokonano wyboru wyłącznie tekstów narracyjnych. Za kryterium przyjęto pochodzenie tekstów. Użyto tych, których źródłem był Project Gutenberg [62], a następnie potwierdzono wybór dodatkową ręczną selekcją. Finalna łączna ilość wykorzystanych dokumentów wyniosła 259 dla QuALITY (4923 pytania, 2606 pytania trudne) i 706 dla NarrativeQA (21894 pytania). Dodatkową zmianą związaną z datasetem QuALITY była rezygnacja z podejścia polegającego na wyborze właściwej odpowiedzi spośród podanych opcji. Zamiast tego poprawna opcja traktowana była jako *golden answer*, do której porównywana jest odpowiedź wygenerowana przez model. Pozwoliło to lepiej ocenić rzeczywistą adekwatność i trafność wykorzystanych kontekstów i sumaryzacji (w wypadku jedynie wyboru odpowiedzi trafność często jest zawyżona).

4.2. Szczegóły implementacji benchmarków

Benchmarki zaimplementowano w wykorzystaniem wspomnianego frameworka RAGAS, przy użyciu języka Python (v3.12) oraz notatników Jupyter, co pozwala na ich łatwe

uruchomienie na różnych środowiskach. Wystarczy w tym celu zainstalować podstawowe zależności zgodnie z plikiem *requirements.txt*, a następnie uruchomić wybrany skrypt bezpośrednio z dostarczonego pliku Jupyter.

Struktura skryptów za każdym razem jest podobna:

1. Załadowanie informacji konfiguracyjnych z predefiniowanych zmiennych (takich jak *OPENAI_API_KEY* - dla wskazania klucza dostępu OpenAI) umieszczonych w plikach *.env*
2. Iteracyjne wczytanie danych z datasetu i wywołanie budowy grafu lub wczytanie zapisanego wcześniej grafu z podanego pliku. Do wczytywania danych wykorzystana jest biblioteka Pandas, dzięki której dane strukturyzowane są w postaci tzw. ramek danych (*dataframe*), co ułatwia dalsze operacje.
3. Budowa przypadków testowych z zastosowaniem wczytanego datasetu. Każdy przypadek testowy składa się z trzech pól: *question*, *ground_truth* i *answer*. Dwa pierwsze uzupełniane są od razu, zgodnie z danymi zebranymi w datasecie.
4. W następnej kolejności do modelu operującego na grafie kierowany jest prompt, z prośbą o odpowiedź na pytanie z zadanego przypadku testowego. Uzyskana odpowiedź uzupełnia pole *answer*.
5. Przygotowane przypadki testowe konwertowane są do obiektu klasy *Dataset* biblioteki Apache Arrow, obsługiwanej przez framework RAGAS.
6. Uzyskany obiekt przekazywany jest następnie do metody *evaluate* w RAGAS, razem z oczekiwanymi metrykami (*answer_correctness*, *answer_similarity*).
7. Wynikiem jest tabelka zawierająca wynik obu metryk dla każdego przypadku testowego. W większości wypadków jako ostateczny rezultat traktowane są średnie metryk dla podanego zbioru.

Przykładowy skrypt do uruchomienia benchmarków może mieć postać przedstawioną poniżej. Można w nim prześledzić najważniejsze wymienione elementy (ze względów bezpieczeństwa pominięto wczytywanie klucza dostępu). W linii 81. wczytywany jest plik zawierający źródłowe teksty, następnie dla każdego z nich budowany jest graf (linie 91-93). Po zbudowaniu grafu wykorzystywany jest on jako źródło dla RAG, a pytania wraz z odpowiedziami i oczekiwanymi odpowiedziami dołączane są do przypadków testowych (linie 94-103). Finalnie uzyskane przypadki oceniane są za pomocą wybranych metryk frameworka RAGAS (linie 109-112):

Listing 5. Przykładowy skrypt do uruchomienia benchmarków

```
20 from datasets import Dataset
21 import pandas as pd
22 from ragas import evaluate
23 from ragas.metrics import answer_correctness, answer_similarity
24
25 (...)
26
```

```
df = pd.read_json(file_path, lines=True)

tests_cases = {
    "question": [],
    "answer": [],
    "ground_truth": [],
}

for index, row in df.iterrows():
    RA = rn.RetrievalAugmentation(
        config=rn.RetrievalAugmentationConfig(qa_model=GPT3TurboQAModel()),
    )
    RA.add_documents(row["article"])
    questions = row["questions"]

    for question in questions:
        tests_cases["question"].append(question["question"])
        tests_cases["answer"].append(RA.answer_question(
            question=f'{question["question"]}.', top_k=5
        ))
        tests_cases["ground_truth"].append(
            question["options"][question["gold_label"]-1]
        )

from datasets import Dataset
from ragas import evaluate
from ragas.metrics import answer_correctness, answer_similarity

dataset = Dataset.from_dict(tests_cases)
score = evaluate(dataset, metrics=[answer_correctness, answer_similarity])
df = score.to_pandas()
df["answer_correctness"].mean(), df["answer_similarity"].mean()
```

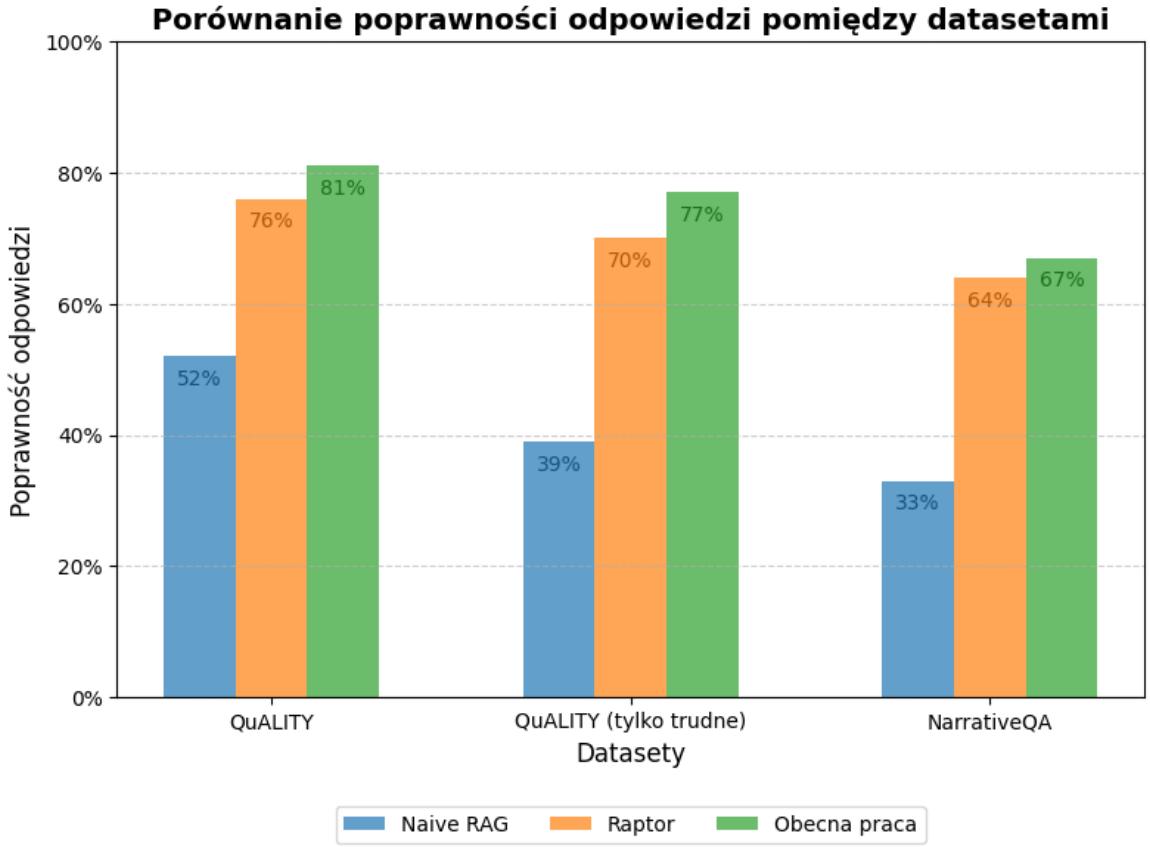
Do wygenerowania wykresów dla uzyskanych danych wykorzystano bibliotekę matplotlib. Pełen kod skryptów do benchmarków oraz do generowania wykresów dołączony jest do całości kodu projektu.

4.3. Analiza wyników

4.3.1. Poprawność odpowiedzi

Podstawowym przeprowadzonym eksperymentem było porównanie poprawności odpowiedzi wyliczanej według strategii opisanej powyżej dla trzech rozwiązań: naiwnego RAG, RAPTOR oraz proponowanego rozwiązania. Należy podkreślić, że RAPTOR na moment pisania tych słów (wrzesień 2024) jest ustalonym *state-of-the-art* dla datasetu QuALITY [63] (uzyskane wyniki różnią się w stosunku do wyników na liście rankingowej ze względu na sposób pomiaru; w tej pracy zgodnie z przyjętą metodologią badane jest podo-

bieństwo wygenerowanej odpowiedzi do odpowiedzi poprawnej, a nie wybór konkretnej odpowiedzi). Uzyskano następujące wyniki:



Rysunek 4.1. Porównanie poprawności odpowiedzi pomiędzy datasetami (wykres)

Dataset	Naiwny RAG	Raptor	Obecna praca
QuALITY	52%	76%	81%
QuALITY (tylko trudne)	39%	70%	77%
NarrativeQA	33%	64%	67%

Tabela 4.1. Porównanie poprawności odpowiedzi pomiędzy datasetami

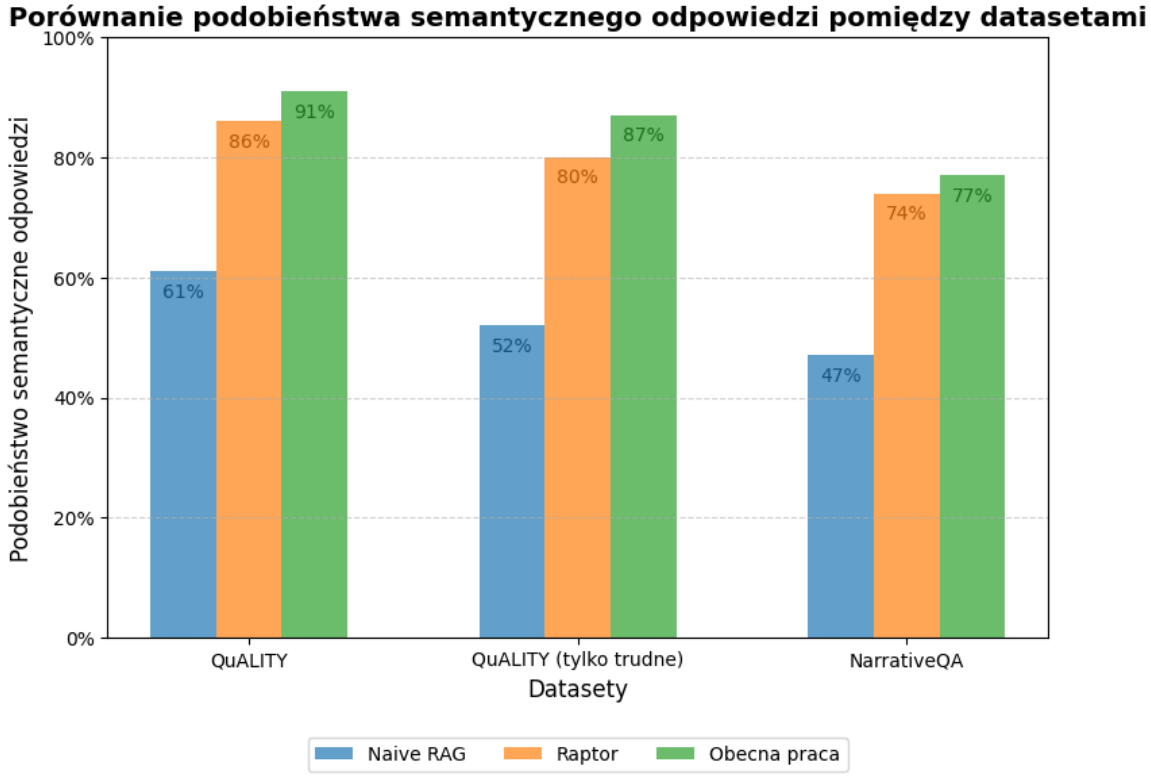
Dla każdego z badanych datasetów uzyskano poprawę względem rozwiązań bazowych. W wypadku datasetu QuALITY prezentowane rozwiązanie uzyskało 91% poprawności. Jest to rezultat o 29 p.p. lepszy od naiwnego RAG (poprawność 61%), oraz o 5 p.p. lepszy od rozwiązania RAPTOR (poprawność 86%). Dla podzbioru pytań trudnych rozwiązanie uzyskało 87% poprawności. W stosunku do innych rozwiązań to wynik lepszy o odpowiednio 38 p.p. od naiwnego RAG (poprawność 52%) i o 7 p.p. względem RAPTOR (poprawność 80%). Dla zbioru NarrativeQA wynik rozwiązania wyniósł 67%, o 34 p.p. lepiej niż naiwny RAG (33%) i o 3 p.p. lepiej niż RAPTOR (64%).

Warto zwrócić uwagę na wartości poprawności odpowiedzi dla poszczególnych datasetów. Najwyższe wartości uzyskano dla całości QuALITY, następnie dla trudnego podzbioru

z tego samego datasetu, a na samym końcu sytuuje się NarrativeQA. Tendencje te są dostrzegalne także w pozostałych eksperymentach. Wynikają one głównie z tego, że NarrativeQA operuje głównie na bardzo długich tekstach, często wielokrotnie dłuższych niż QuALITY. Z tego powodu znalezienie właściwych informacji wraz z szerokim kontekstem jest w tym wypadku trudniejsze. Pokreślić jednak należy, że wartości na poziomie powyżej 60% uznać trzeba w tym wypadku za bardzo dobre.

4.3.2. Poprawność semantyczna odpowiedzi

Kolejnym przeprowadzonym testem był pomiar i porównanie poprawności semantycznej wygenerowanych odpowiedzi dla każdego z rozwiązań. Wyniki przedstawione są na wykresie:



Rysunek 4.2. Porównanie podobieństwa semantycznego odpowiedzi pomiędzy datasetami (wykres)

Dataset	Naive RAG	Raptor	Obecna praca
QuALITY	61%	86%	91%
QuALITY (tylko trudne)	52%	80%	87%
NarrativeQA	47%	74%	77%

Tabela 4.2. Porównanie podobieństwa semantycznego odpowiedzi pomiędzy datasetami

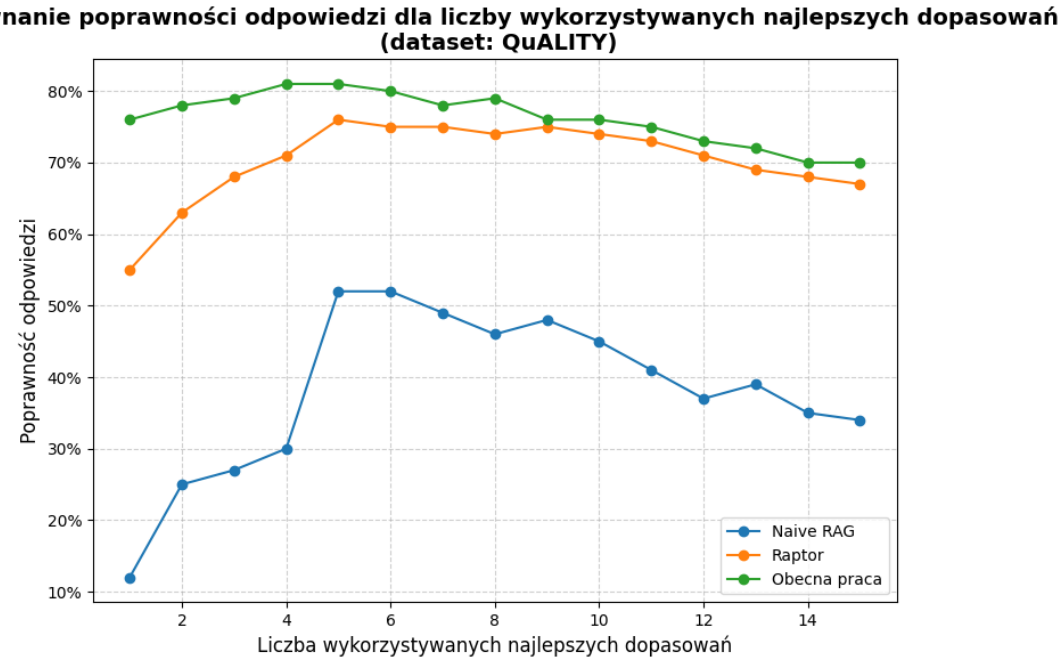
Również w tym wypadku widać wyraźną poprawę. Dla QuALITY proponowane rozwiązanie uzyskało 91%, czyli wynik o 30 p.p. lepszy niż naiwne RAG (61%) i o 5 p.p. lepszy

niż RAPTOR (86%). Dla podzbioru trudnych z QuALITY wynik to 87%, o 35 p.p. więcej niż w wypadku naiwnego RAG (52%) i 7 p.p. więcej niż RAPTOR (80%). Dla NarativeQA podobieństwo semantyczne jest na poziomie 77%, o 30 p.p. więcej niż dla naiwnego RAG (47%) i o 3 p.p. więcej niż dla RAPTOR (74%).

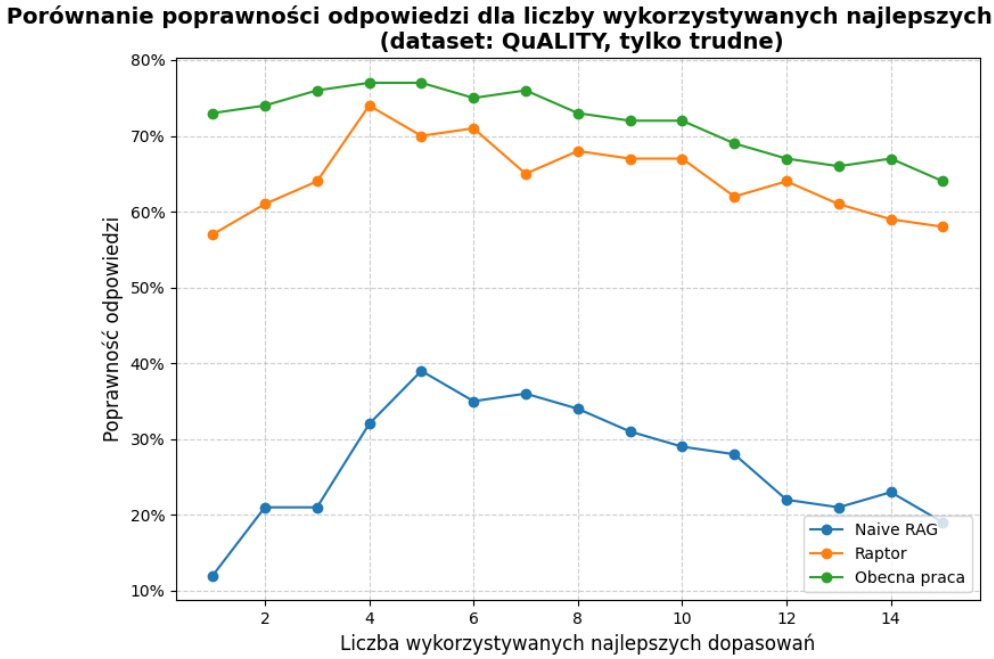
Pokreślić należy, że wartości dla podobieństwa semantycznego są generalnie wyższe niż dla poprawności odpowiedzi. Wynika to z tego, że fakty w odpowiedziach wygenerowanej i oczekiwanej nie są sprawdzane wprost, a nawet podobieństwo w sformułowaniu odpowiedzi może podwyższać wynik podobieństwa cosinusowego. Z tego względu, tak jak wspomniano wcześniej, w kolejnych eksperymentach posłużono się poprawnością odpowiedzi jako metryką bazową, najlepiej oceniającą faktyczną jakość rozwiązania.

4.3.3. Wpływ liczby wykorzystanych rezultatów na poprawność odpowiedzi

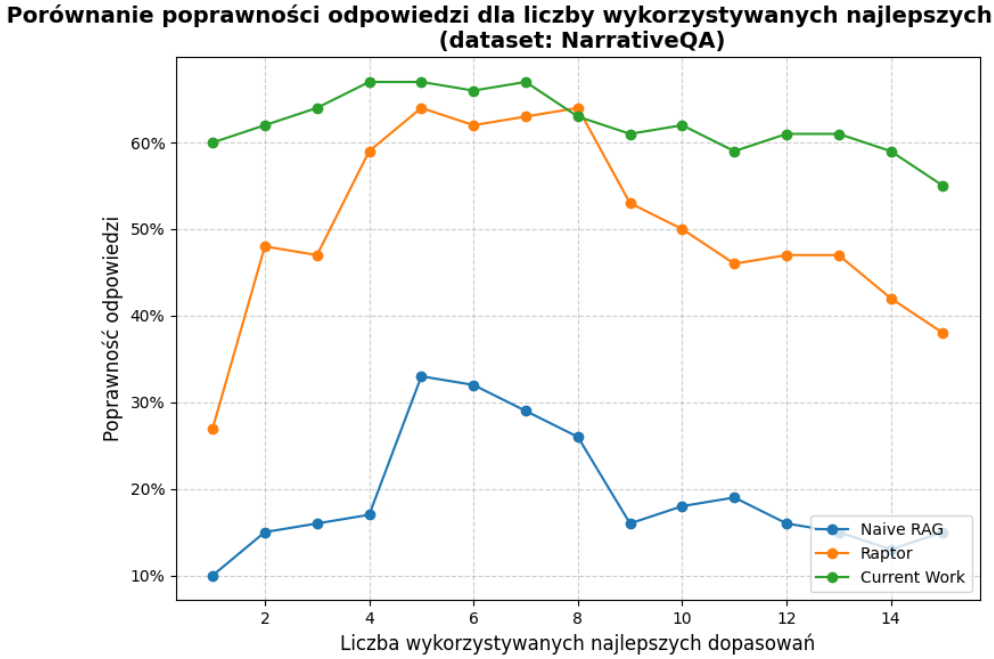
W kolejnym eksperymencie sprawdzono wpływ ilości wybieranych najlepiej dopasowanych węzłów (*top k*) wstrzykiwanych do kontekstu na finalną jakość odpowiedzi, w celu wyboru najlepszej wartości. We wszystkich przypadkach można prześledzić podobne tendencje, wynikające z uwarunkowań dużych modeli. Szczegółowe wyniki przedstawiono na rysunkach 4.3, 4.4 i 4.5.



Rysunek 4.3. Wpływ liczby wykorzystanych rezultatów na poprawność odpowiedzi - QuALITY



Rysunek 4.4. Wpływ liczby wykorzystanych rezultatów na poprawność odpowiedzi - QuALITY (trudne)



Rysunek 4.5. Wpływ liczby wykorzystanych rezultatów na poprawność odpowiedzi - NarrativeQA

Na wszystkich wykresach widać, że najlepsze wyniki osiągane są przy używaniu 4-6 najlepiej dopasowanych węzłów. Średnio najlepszą wartością jest używanie 5 najlepszych rezultatów, i tą wartością posługiwano się w pozostałych eksperymentach.

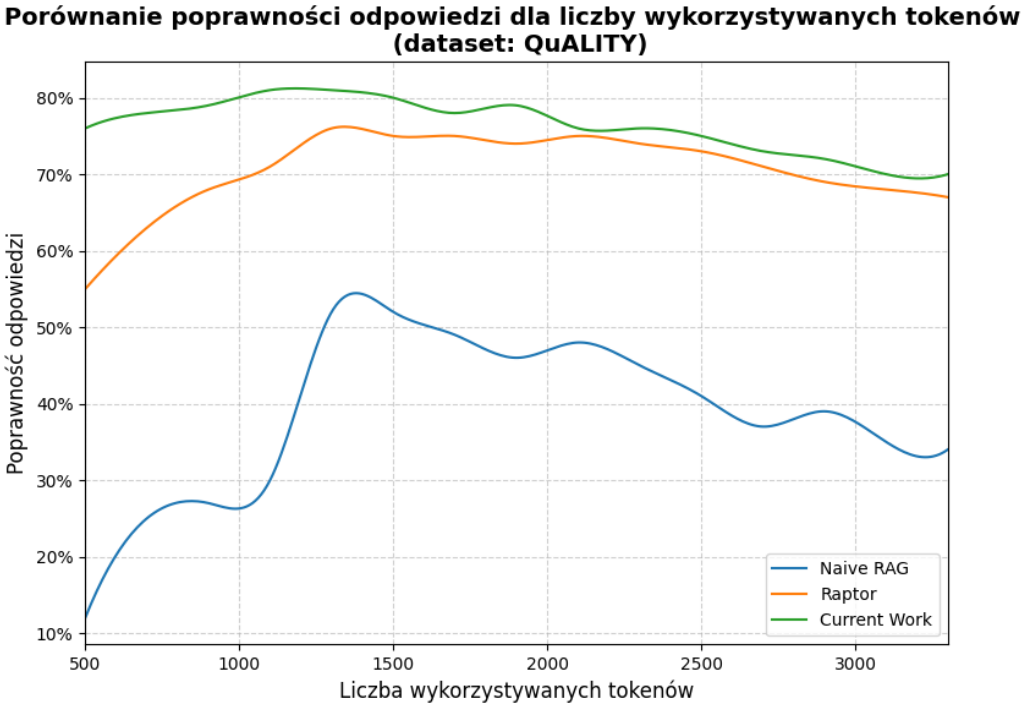
Słabsze rezultaty osiągane poniżej tych wartości dają się łatwo wytłumaczyć, gdyż model nie otrzymuje wystarczającej ilości potrzebnych informacji, i z tego względu nie jest w stanie udzielić poprawnej odpowiedzi. Warto jednak zwrócić uwagę, że prezentowane rozwiązanie jest stosunkowo odporne na zmniejszanie ilości dołączanych dopasowań, uzyskując odpowiednio 76% poprawności dla datasetu QuALITY, 71% dla podzbioru trudnych, i 60% dla NarrativeQA przy tylko jednym najlepszym dopasowaniu. Wynika to z przyjętej struktury streszczeń, dzięki czemu istnieje prawdopodobieństwo, że sens odpowiedzi jest już zawarty w którymś z węzłów.

Trzeba podkreślić, że wyniki uzyskane przy jednym lub dwóch węzłach znacznie przewyższają wyniki osiągane przez inne rozwiązania. Z tego względu proponowane podejście może być szczególnie przydatne w sytuacjach, gdy ważna jest efektywność kosztowa lub czasowa, gdyż posługiwanie się mniejszą ilością kontekstów pozytywnie wpływa na oba parametry.

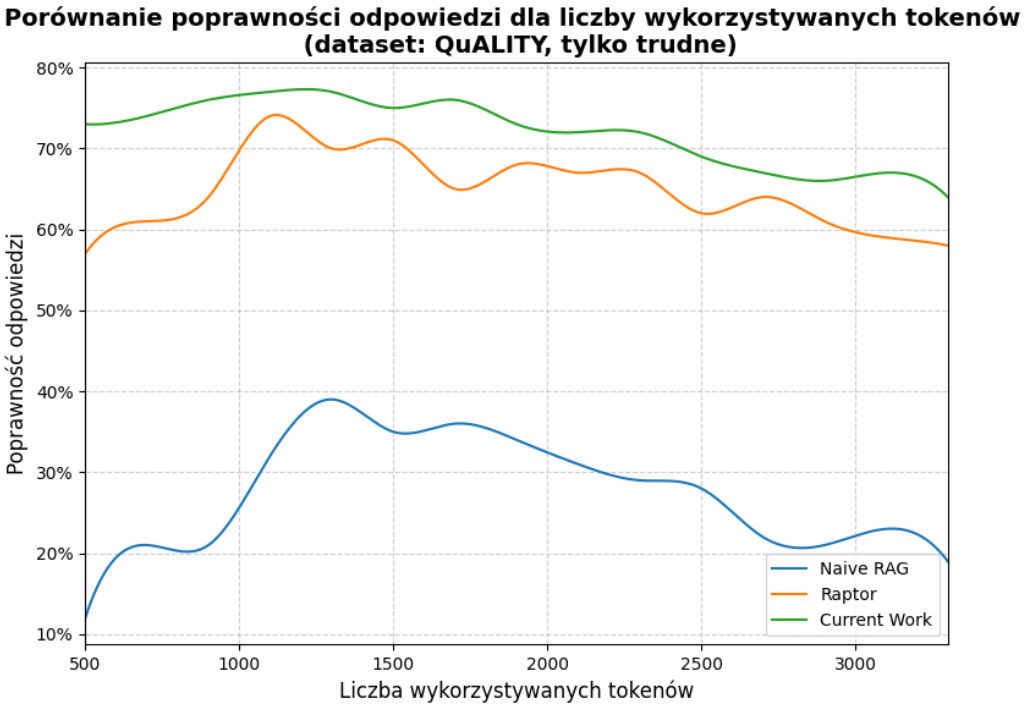
Stopniowe obniżanie się jakości odpowiedzi przy rosnącej ilości używanych najlepiej dopasowanych węzłów wytłumaczyć można wspomnianym wcześniej zjawiskiem zagubienia się w środku (*lost in the middle*) [18]. W wypadku zbyt dużej ilości informacji dostarczonej do modelu, kluczowe fakty są zbyt rozwodnione i trudniejsze do wyekstrahowania i użycia przez model.

4.3.4. Wpływ liczby tokenów na poprawność odpowiedzi

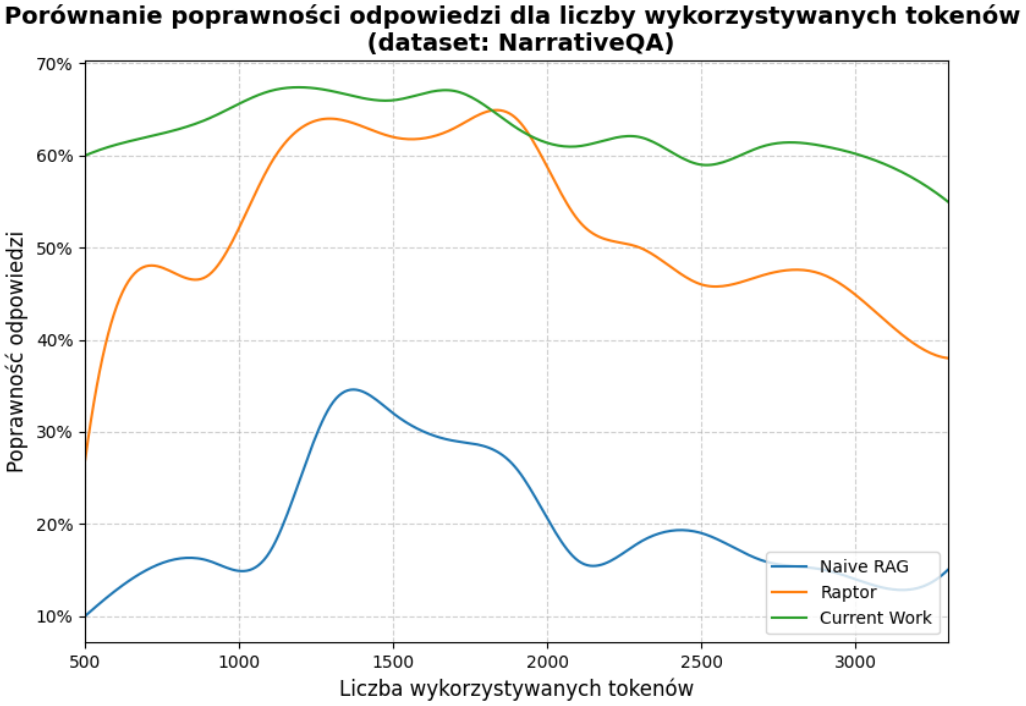
Ściśle powiązaną metryką jest wpływ ilości użytych tokenów na poprawność odpowiedzi. Wpływ ten przedstawiony został na rysunkach 4.6, 4.7 i 4.8.



Rysunek 4.6. Wpływ liczby tokenów na poprawność odpowiedzi - QuALITY



Rysunek 4.7. Wpływ liczby tokenów na poprawność odpowiedzi - QuALITY (trudne)



Rysunek 4.8. Wpływ liczby tokenów na poprawność odpowiedzi - NarrativeQA

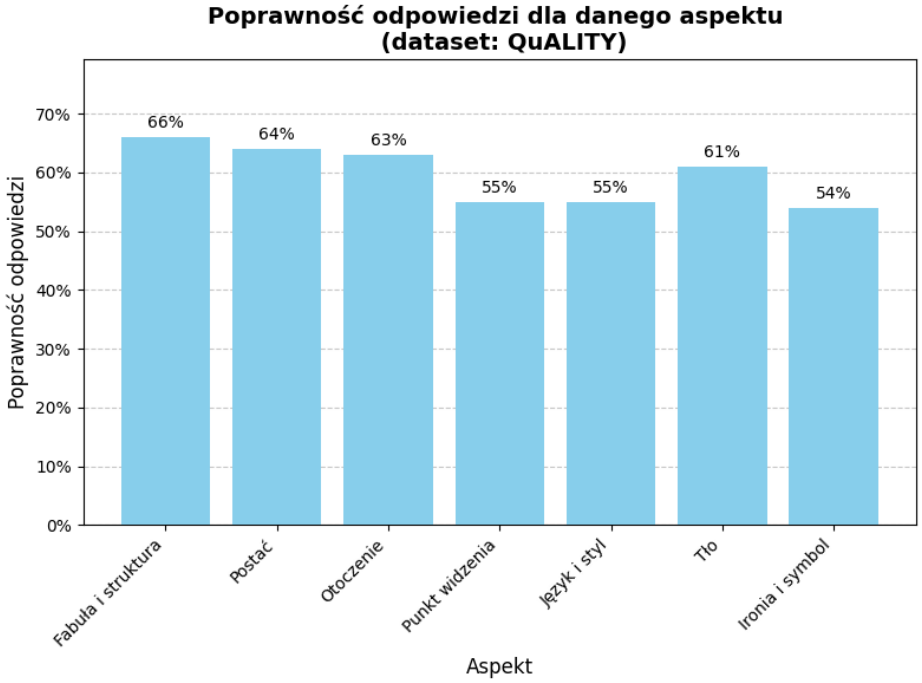
Obserwacje są bliskie obserwacjom wpływu ilości wykorzystywanych najlepszych dopasowań. Najlepsze wyniki uzyskiwane są w przedziale 1000-1700 tokenów.

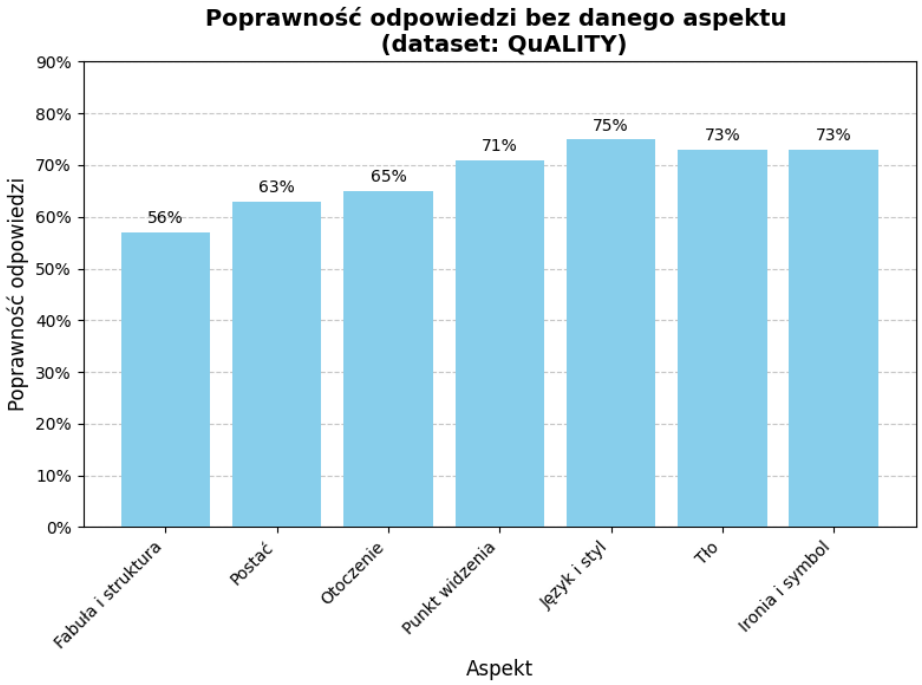
4.3.5. Wpływ poszczególnych aspektów na finalną odpowiedź

Odrębną badaną kwestią był wpływ poszczególnych aspektów na finalną poprawność odpowiedzi. Eksperymentów dokonano w dwóch perspektywach: poprzez sprawdzenie poprawności przy generacji podsumowań tylko dla jednego aspektu, oraz poprawność przy generacji podsumowań z wyłączeniem konkretnego aspektu. Warto zwrócić uwagę, że w wypadku budowy grafu dla tylko jednego aspektu wyniki wciąż potrafią osiągać względnie wysokie wartości (w okolicach 30 - 50%). Wynika to z faktu, że w grafach tych wciąż dostępna jest całość treści we fragmentach, co pozwala na poprawną odpowiedź na niektóre pytania.

Uzyskane wyniki przedstawiono na rysunkach 4.9, 4.10 i 4.11.

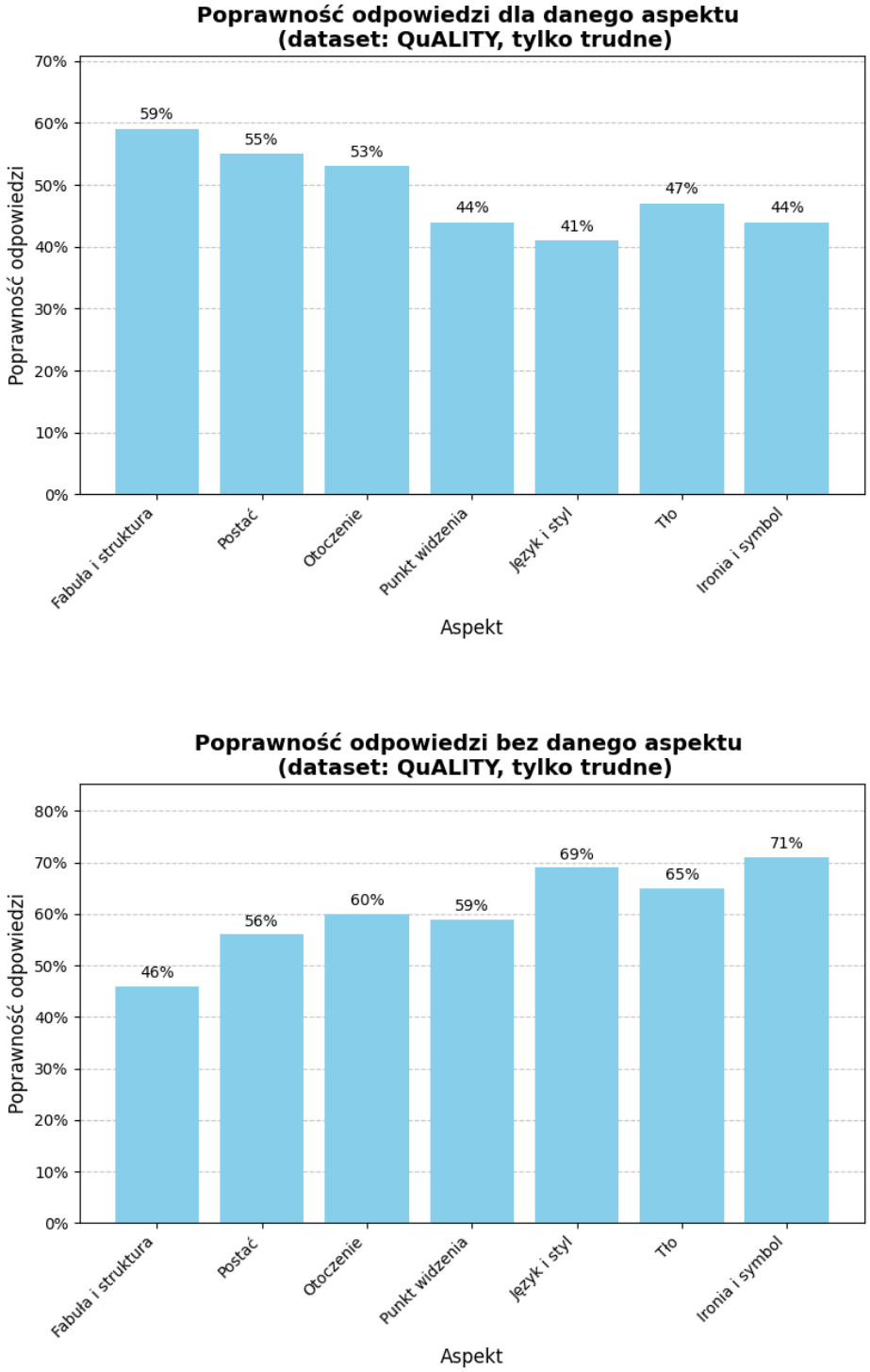
- **Dataset QuALITY**





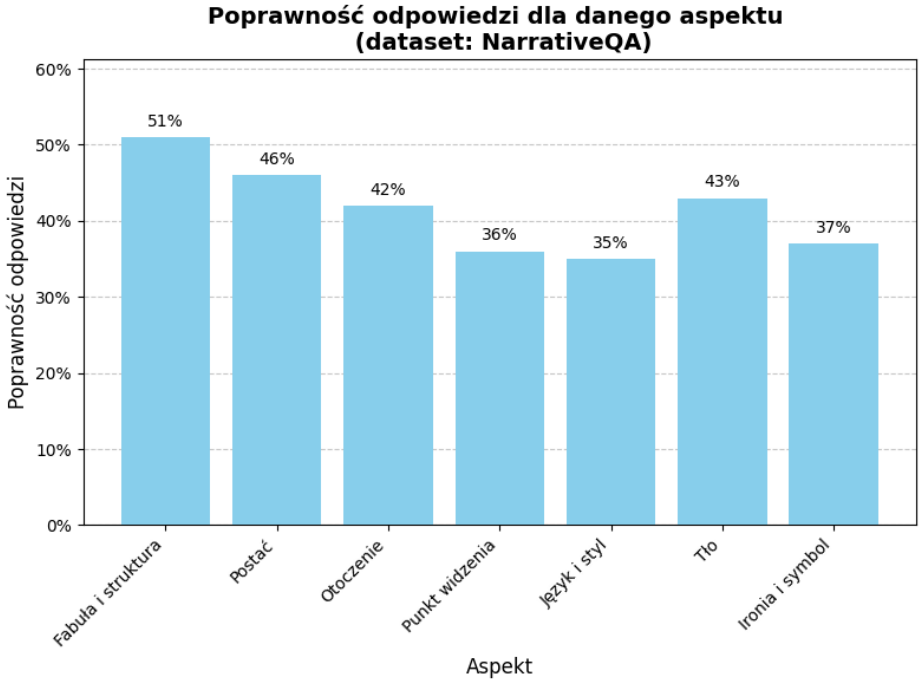
Rysunek 4.9. Dataset QuALITY - poprawność odpowiedzi dla pojedynczych aspektów i z ich wyłączeniem

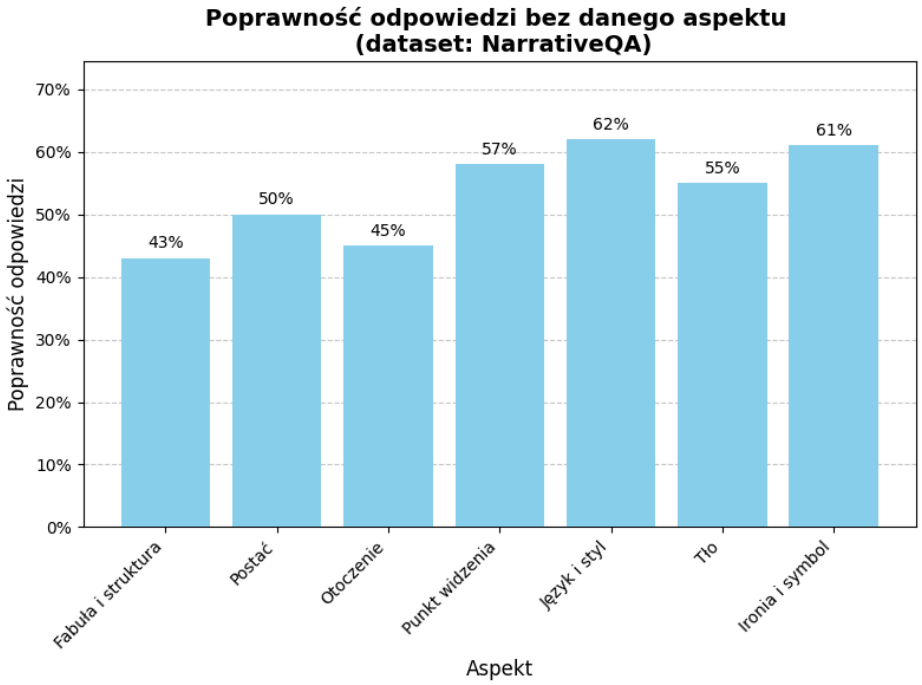
- **Dataset QuALITY (tylko trudne)**



Rysunek 4.10. Dataset QuALITY (tylko trudne) - poprawność odpowiedzi dla pojedynczych aspektów i z ich wyłączeniem

- **Dataset NarrativeQA**





Rysunek 4.11. Dataset NarrativeQA - poprawność odpowiedzi dla pojedynczych aspektów i z ich wyłączeniem

We wszystkich trzech wypadkach dają się prześledzić podobne tendencje, chociaż występujące z różnym nasileniem. Największy wpływ na wzrost poprawności odpowiedzi mają:

- Fabuła i struktura
- Postać
- Otoczenie
- Tło (w mniejszym zakresie)

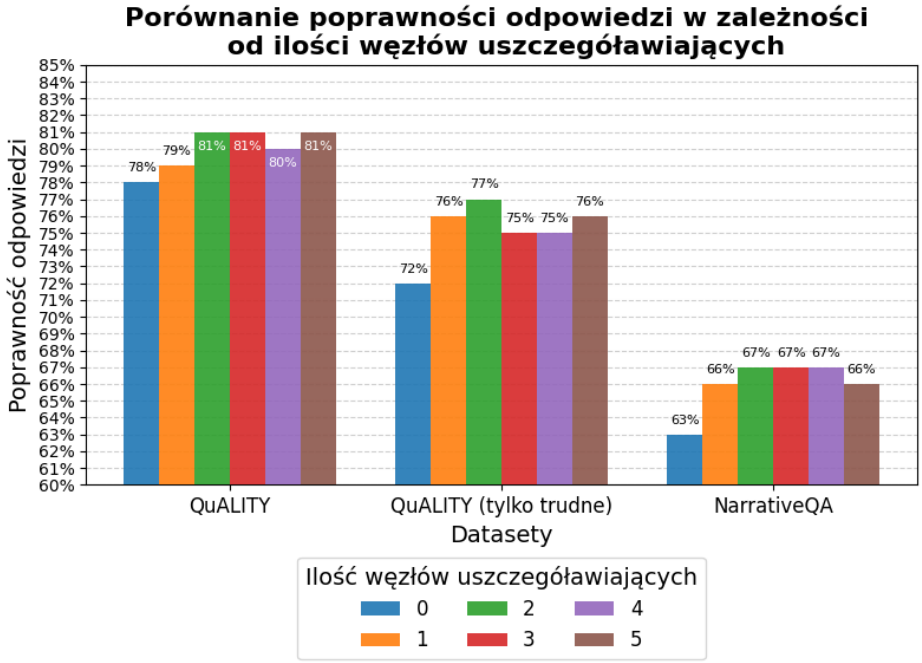
Z kolei najmniejszy:

- Język i styl
- Ironia i symbol

Podkreślić jednak należy, że wykluczenie nawet aspektów o najmniejszym wpływie prowadzi do pogorszenia rozwiązań. Pozwala to potwierdzić hipotezę, że każdy z tych aspektów jest istotnym elementem tekstów narracyjnych (nawet jeśli ich waga jest różna).

4.3.6. Wpływ ilości węzłów uszczegóławiających na jakość odpowiedzi

Analizie poddano również wpływ dodania węzłów uszczegóławiających do każdego z fragmentów bazowego tekstu. Wyniki przedstawiono na rysunku 4.12.



Rysunek 4.12. Porównanie poprawności odpowiedzi w zależności od ilości węzłów uszczegóławiających

Można zaobserwować, że pozytywny skutek mają przede wszystkim dodanie pierwszego węzła (zysk od 1 p.p. do nawet 4 p.p. jeśli chodzi o jakość odpowiedzi), oraz drugiego węzła, choć w tym wypadku wzrost jest już mniejszy (1 - 2 p.p.). Dodawanie kolejnych

węzłów nie przyniosło dalszej poprawy rezultatów, a w kilku wypadkach doprowadziło do regresji.

Zjawisko poprawy rezultatów po dodaniu pierwszego węzła można tłumaczyć bardziej syntetycznym ujęciem treści fragmentu. Dzięki wprowadzonemu rozwiązaniu, wszystkie kluczowe detale są dobrze wyodrębnione, a przez to czytelne i łatwe do wykorzystania przez model podczas generacji odpowiedzi. Poprawa wynikająca z dodania drugiego węzła wynika z różnic w formie kolejnych uszczegółowień. Model ma względną swobodę w jego generacji, a tym samym dwa węzły mogą się różnić strukturą, sposobem prezentacji ważnych detali, a nawet ich doborem. Można to prześledzić na przykładzie:

Tekst orginalny (H. Sienkiewicz: *Ogniem i mieczem* [64])

Więc pan Skrzetuski zaprosił go do kompanii. Zapach pieczonych mięs jął właśnie rozchodzić się od stosu, łechcąc nozdrza i podniebienie. Pachoł wydobył je z żaru i podał na latercynowej misie. Poczęli jeść, a gdy przyniesiono spory worek mołdawskiego wina, uszyty z koźlej skóry, wnet zawiązała się żywa rozmowa.
— Oby nam się szczęśliwie do domu wróciło! — rzekł pan Skrzetuski.
— To waszmość wracasz? Skądże, proszę? — spytał Abdank.
— Z daleka, bo z Krymu.
— A cóżeś waszmość tam robił? Z wykupnym jeździłeś?
— Nie, mości pułkowniku; jeździłem do samego chana.
Abdank nastawił ciekawie ucha.
— Ano to, proszę, w piękną waść wszedłeś komitywę! I z czymże do chana jeździłeś?
— Z listem jaśnie oświeconego księcia Jeremiego.
— To waść posłował! O cóż jegomość książę do chana pisał?
Namiestnik popatrzył bystro na towarzysza.
— Mości pułkowniku — rzekł — zaglądałeś w oczy łotrzykom, którzy cię na arkan ujęli, to twoja sprawa; ale co książę do chana pisał, to ani twoja, ani moja, jeno ich obydwóch.

Pierwsze uszczegółowienie

Pan Skrzetuski zaprosił Abdanka do wspólnego posiłku.
Zapach pieczonego mięsa roznosił się, jedli z latercynowej misy.
Rozmowa o winie mołdawskim, które przyniesiono w worku z koźlej skóry.
Skrzetuski mówił o powrocie do domu i o swojej podróży.
Abdank pyta, skąd Skrzetuski wraca — z Krymu.
Skrzetuski wyjaśnia, że nie był tam z wykupnym, ale z listem od księcia Jeremiego do chana.

Abdank jest ciekawy, o czym książę pisał do chana.
Skrzetuski odpowiada, że to tajemnica między księciem a chanem, nie ich sprawa.

Drugie uszczegółowienie

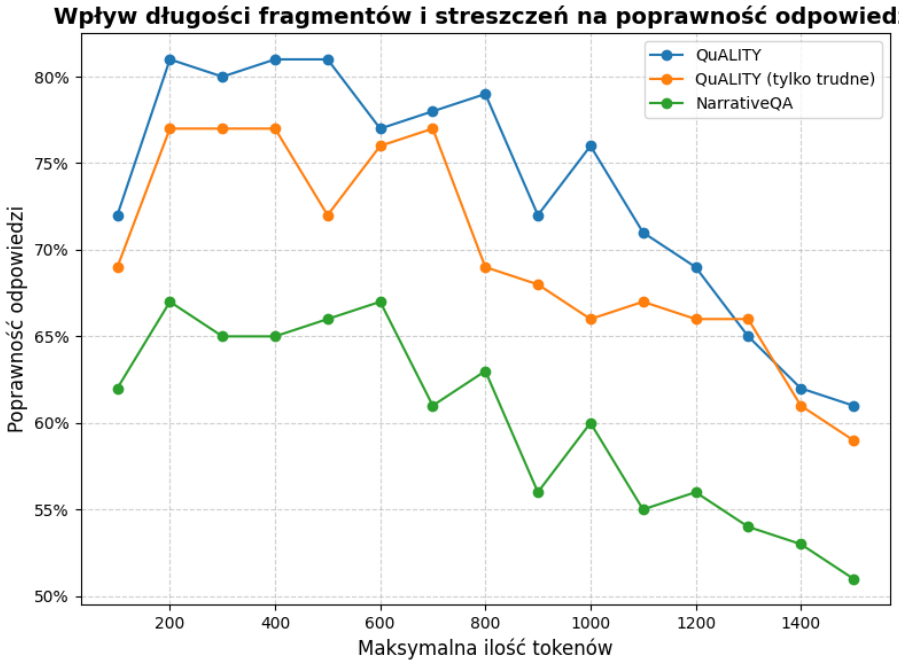
Pan Skrzetuski zaprosił Abdanka na posiłek, jedli pieczone mięso.
Pojawił się worek mołdawskiego wina, co ożywiło rozmowę.
Skrzetuski mówi o powrocie z Krymu.
Abdank pyta, czy Skrzetuski był tam z wykupnym, ale on odpowiada, że jeździł do chana.
Abdank jest ciekawy, co Skrzetuski robił u chana.
Skrzetuski wyjaśnia, że był z listem od księcia Jeremiego.
Abdank pyta o treść listu.
Skrzetuski odpowiada, że to sprawa między księciem a chanem, nie ich.

Wykorzystanie tego zróżnicowania w trakcie wyszukiwania węzłów pasujących do zapytania ma zgodnie z uzyskanymi wynikami korzystne skutki.

Dodanie trzeciego i następnych węzłów nie wzmacnia tego jednak efektu. Obserwacje wskazują, że kolejne dodawane węzły stają się do siebie zbyt podobne, a we fragmentach jest zbyt mała ilość szczegółów, by ujęcie ich w odmienny sposób wciąż było poznawczo wartościowe. Zdarza się również, że istotne pogorszenie jakości zachodzi w fazie wyboru węzłów do kontekstu. Dla niektórych pytań rezultatem poszukiwania najbliższych reprezentacji wektorowych jest dobór głównie spośród węzłów z uszczegółowieniami, nawet jeśli są one do siebie bardzo zbliżone. Rezultaty te mogą ograniczyć ilość wyszukanych i zwróconych do kontekstu węzłów innego typu, co finalnie ma niekorzystny wpływ na poprawność odpowiedzi modelu.

4.3.7. Wpływ długości fragmentów i streszczeń na jakość odpowiedzi

Badaniu poddano również również wpływ długości fragmentów i streszczeń na poprawność odpowiedzi. Wyniki można prześledzić na rysunku 4.13.



Rysunek 4.13. Porównanie poprawności odpowiedzi w zależności od długości fragmentów i streszczeń

Widać wyraźnie, że najlepsze wyniki uzyskano w zakresie maksimów od 200 do 600 tokenów. Powyżej tej wartości następuje stopniowy spadek poprawności odpowiedzi. Proces ten ponownie można wytłumaczyć narastaniem zjawiska zagubienia w środku - przy zbyt długich kontekstach poszukiwane szczegóły rozmywają się w całości uzyskanego tekstu i są trudniejsze do wykorzystania przez model.

Średnio najlepsze rezultaty uzyskano przy maksymalnej wartości tokenów ustawionej na 200, i z tego względu w innych eksperymentach posługiwano się tą wartością.

5. Podsumowanie

Przedstawiona praca wprowadza nowe podejście do tematu organizacji wiedzy dla Retrieval Augmented Generation dla tekstów narracyjnych, dzięki grafowej organizacji wiedzy rozszerzając możliwości oferowane przez istniejące obecnie narzędzia.

Wstępem do realizacji tego zadania stała się analiza istniejącego stanu wiedzy, z szeroko zakrojonym przeglądem literatury. Dzięki temu etapowi udało się zidentyfikować i opisać podstawowe metodologie działania z dużymi modelami językowymi oraz frameworkiem Retrieval Augmented Generation. Analizie poddano również kluczowe komponenty projektowanych obecnie rozwiązań, od omówienia architektury samych dużych modeli językowych, poprzez prześledzenie źródeł i rozwoju Retrieval Augmented Generation, aż po wprowadzenie grafów wiedzy w kontekście RAG.

Osobnym przedmiotem analizy stały się teksty narracyjne i wyróżnienie istotnych aspektów, stanowiących różne perspektywy odbioru i rozumienia tekstu. W tym celu dokonano przeglądu reprezentatywnych prac z zakresu teorii literatury i selekcji takiego zestawu aspektów, który zgodny jest z generalnym naukowym konsensusem. Pozytywne wyniki eksperymentów dowodzą trafności dokonanego wyboru.

Oba te źródła stały się podstawą dla zaproponowanego rozwiązania. Finalnie więc praca dostarcza także kompletną implementację narzędzi oraz dokładny opis metod budowy grafu, który pozwala w taki sposób zorganizować informacje z tekstów narracyjnych, by rezultaty Retrieval Augmented Generation były lepsze niż rezultaty osiągane przez istniejące rozwiązania. Głównymi wyróżnikami konstruowanego grafu są wielostopniowa struktura, pozwalająca ująć sensy tekstu na rosnących poziomach generalizacji, oraz wieloaspektowość, dzięki której można odwzorować różne warstwy znaczeniowe dzieła literackiego.

Ważnym elementem pracy było również przetestowanie wpływu różnych parametrów (takich jak długość pojedynczego fragmentu, całkowita długość kontekstu, ilość wykorzystywanych najlepszych dopasowań etc) na poprawność osiąganych rezultatów. Dzięki przeprowadzonym eksperymentom ustalono, jaki zestaw parametrów daje najlepsze wyniki. W wypadkach, kiedy było to możliwe, dodano także poszerzone wyjaśnienia teoretyczne, dlaczego określone zmiany w parametrach wpływają pozytywnie lub negatywnie na finalny wynik.

Dodatkową kontrybucją stało się dostosowanie istniejących datasetów takich jak QuALITY i NarrativeQA i selekcja tekstów, które określić można jako narracyjne. Razem z narzędziem dostarczono cały zestaw skryptów do pomiaru jakości odpowiedzi w kontekście wspomnianych datasetów, gotowych do użycia w razie podjęcia dalszych prac. Cały kod związany z pracą oraz wszystkie przygotowane benchmarki są dostępne i mogą stać się punktem startowym do kontynuacji badań.

Rozwój prac w tym zakresie może objąć próby wykorzystania dedykowanych modeli do sumaryzacji, specjalnie wytrenowanych pod kątem rozpoznawania konkretnych aspektów

tekstów narracyjnych. Kolejnym obszarem poszukiwań może stać się także zbadanie innych możliwości trawersowania grafu; istnieje szansa, że odpowiednie heurystyki w tym zakresie mogłyby poprawić jakość rozwiązania. Finalnie, sama metoda budowy grafu potencjalnie może być przeniesiona na długie teksty innych rodzajów niż teksty narracyjne. Ten wypadek wymaga jedynie identyfikacji kluczowych aspektów dla wybranych klas tekstów, sama metoda powinna działać bez istotnych przekształceń.

Bibliografia

[1] ChatGPT, *ChatGPT*, Dostęp zdalny (14.09.2024): https://openai.com/chatgpt/, 2024.

[2] R. Gozalo-Brizuela i E. C. Garrido-Merchán, *A survey of Generative AI Applications*, 2023. arXiv: 2306.02781 [cs.LG]. adr.: https://arxiv.org/abs/2306.02781.

[3] N. Muennighoff, N. Tazi, L. Magne i N. Reimers, *MTEB: Massive Text Embedding Benchmark*, 2023. arXiv: 2210.07316 [cs.CL]. adr.: https://arxiv.org/abs/2210.07316.

[4] W.-L. Chiang, L. Zheng, Y. Sheng i in., *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*, 2024. arXiv: 2403.04132 [cs.AI]. adr.: https://arxiv.org/abs/2403.04132.

[5] H. Face, *Open LLM Leaderboard - About*, Dostęp zdalny (21.07.2024): https://huggingface.co/docs/leaderboards/open_llm_leaderboard/about, 2024.

[6] J. Wei, X. Wang, D. Schuurmans i in., *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, 2023. arXiv: 2201.11903 [cs.CL]. adr.: https://arxiv.org/abs/2201.11903.

[7] S. Yao, D. Yu, J. Zhao i in., *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*, 2023. arXiv: 2305.10601 [cs.CL]. adr.: https://arxiv.org/abs/2305.10601.

[8] M. Besta, N. Blach, A. Kubicek i in., "Graph of Thoughts: Solving Elaborate Problems with Large Language Models", *Proceedings of the AAAI Conference on Artificial Intelligence*, t. 38, nr. 16, s. 17 682–17 690, mar. 2024, ISSN: 2159-5399. DOI: 10.1609/aaai.v38i16.29720. adr.: http://dx.doi.org/10.1609/aaai.v38i16.29720.

[9] A. Vaswani, N. Shazeer, N. Parmar i in., "Attention is all you need", *Advances in neural information processing systems*, t. 30, 2017.

[10] T. B. Brown, B. Mann, N. Ryder i in., "Language models are few-shot learners", *Advances in Neural Information Processing Systems*, t. 33, s. 1877–1901, 2020.

[11] OpenAI, *ChatGPT*, 2022. adr.: https://openai.com/chatgpt.

[12] CNBC, *ChatGPT is the fastest-growing app in history*, 2023. adr.: https://www.cnbc.com/2023/01/31/chatgpt-is-the-fastest-growing-app-in-history-ubs.html.

[13] G. Copilot, *GitHub Copilot*, Dostęp zdalny (14.09.2024): https://github.com/features/copilot, 2024.

[14] J. Ferrer, *An Introductory Guide to Fine-Tuning LLMs*, Dostęp zdalny (12.07.2024): https://www.datacamp.com/tutorial/fine-tuning-large-language-models, 2024.

[15] S. Das, *Fine Tune Large Language Model (LLM) on a Custom Dataset with QLoRA*, Dostęp zdalny (12.07.2024): https://dassum.medium.com/fine-tune-large-language-model-llm-on-a-custom-dataset-with-qlora-fb60abdeba07, 2024.

[16] P. Lewis, E. Perez, A. Piktus i in., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, 2021. arXiv: 2005.11401 [cs.CL]. adr.: https://arxiv.org/abs/2005.11401.

[17] D. Naukowy. "Ogłoszono laureatów Nagród Fundacji na rzecz Nauki Polskiej". (2024).

[18] N. F. Liu, K. Lin, J. Hewitt i in., *Lost in the Middle: How Language Models Use Long Contexts*, 2023. arXiv: 2307.03172 [cs.CL]. adr.: https://arxiv.org/abs/2307.03172.

[19] P. E. Guide, *Retrieval Augmented Generation (RAG) for LLMs*, Dostęp zdalny (12.07.2024): https://www.promptingguide.ai/research/rag, 2024.

[20] T. Mikolov, K. Chen, G. Corrado i J. Dean, *Efficient Estimation of Word Representations in Vector Space*, 2013. arXiv: 1301.3781 [cs.CL]. adr.: https://arxiv.org/abs/1301.3781.

[21] J. Pennington, R. Socher i C. D. Manning, "Glove: Global Vectors for Word Representation.", w *EMNLP*, t. 14, 2014, s. 1532–1543.

[22] A. Joulin, E. Grave, P. Bojanowski i T. Mikolov, *Bag of Tricks for Efficient Text Classification*, 2016. arXiv: 1607.01759 [cs.CL]. adr.: https://arxiv.org/abs/1607.01759.

[23] M. E. Peters, M. Neumann, M. Iyyer i in., *Deep contextualized word representations*, 2018. arXiv: 1802.05365 [cs.CL]. adr.: https://arxiv.org/abs/1802.05365.

[24] J. Devlin, M.-W. Chang, K. Lee i K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. arXiv: 1810.04805 [cs.CL]. adr.: https://arxiv.org/abs/1810.04805.

[25] J. Howard i S. Ruder, *Universal Language Model Fine-tuning for Text Classification*, 2018. arXiv: 1801.06146 [cs.CL]. adr.: https://arxiv.org/abs/1801.06146.

[26] Y. Liu, M. Ott, N. Goyal i in., *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, 2019. arXiv: 1907.11692 [cs.CL]. adr.: https://arxiv.org/abs/1907.11692.

[27] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov i Q. V. Le, *XLNet: Generalized Autoregressive Pretraining for Language Understanding*, 2020. arXiv: 1906.08237 [cs.CL]. adr.: https://arxiv.org/abs/1906.08237.

[28] Y. Gao, Y. Xiong, X. Gao i in., *Retrieval-Augmented Generation for Large Language Models: A Survey*, 2024. arXiv: 2312.10997 [cs.CL]. adr.: https://arxiv.org/abs/2312.10997.

[29] Weaviate, *Weaviate*, Dostęp zdalny (14.07.2024): https://weaviate.io/, 2024.

[30] Pinecone, *Pinecone*, Dostęp zdalny (14.07.2024): https://www.pinecone.io/, 2024.

[31] Elasticsearch, *Elasticsearch*, Dostęp zdalny (14.07.2024): https://www.elastic.co/elasticsearch/, 2024.

[32] PostgreSQL, *PostgreSQL*, Dostęp zdalny (14.07.2024): https://www.postgresql.org/, 2024.

[33] pgvector, *pgvector*, Dostęp zdalny (14.07.2024): https://github.com/pgvector/pgvector, 2024.

[34] Y. A. Malkov i D. A. Yashunin, *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*, 2018. arXiv: 1603.09320 [cs.DS]. adr.: https://arxiv.org/abs/1603.09320.

[35] C. Long, Y. Liu, C. Ouyang i Y. Yu, *Bailicai: A Domain-Optimized Retrieval-Augmented Generation Framework for Medical Applications*, 2024. arXiv: 2407.21055 [cs.CL]. adr.: https://arxiv.org/abs/2407.21055.

[36] C. Xiao, H. Zhong, Z. Guo i in., *CAIL2018: A Large-Scale Legal Dataset for Judgment Prediction*, 2018. arXiv: 1807.02478 [cs.CL]. adr.: https://arxiv.org/abs/1807.02478.

[37] X. Peng i L. Chen, *Athena: Retrieval-augmented Legal Judgment Prediction with Large Language Models*, 2024. arXiv: 2410.11195 [cs.CL]. adr.: https://arxiv.org/abs/2410.11195.

[38] Y. Pu, Z. He, T. Qiu, H. Wu i B. Yu, *Customized Retrieval Augmented Generation and Benchmarking for EDA Tool Documentation QA*, 2024. arXiv: 2407.15353 [cs.CL]. adr.: https://arxiv.org/abs/2407.15353.

[39] K. Sawarkar, A. Mangal i S. R. Solanki, *Blended RAG: Improving RAG (Retriever-Augmented Generation) Accuracy with Semantic Search and Hybrid Query-Based Retrievers*, 2024. arXiv: 2404.07220 [cs.IR]. adr.: https://arxiv.org/abs/2404.07220.

[40] S. Zerhoudi i M. Granitzer, *PersonaRAG: Enhancing Retrieval-Augmented Generation Systems with User-Centric Agents*, 2024. arXiv: 2407.09394 [cs.IR]. adr.: https://arxiv.org/abs/2407.09394.

[41] J. Barrasa, *What Is a Knowledge Graph?*, Dostęp zdalny (03.07.2024): https://neo4j.com/blog/what-is-knowledge-graph/, 2023.

[42] Neo4j, Inc., *Neo4j (5.0 Release)*, Dostęp zdalny (03.07.2024): https://neo4j.com/blog/announcing-neo4j-5-graph-database/, 2022.

[43] NebulaGraph Inc., *NebulaGraph (3.8.0)*, Dostęp zdalny (03.07.2024): https://docs.nebula-graph.io/3.8.0/1.introduction/1.what-is-nebula-graph/, 2024.

[44] S. Ji, S. Pan, E. Cambria, P. Marttinen i P. S. Yu, "A Survey on Knowledge Graphs: Representation, Acquisition, and Applications", *IEEE Transactions on Neural Networks and Learning Systems*, t. 33, nr. 2, s. 494–514, lut. 2022, ISSN: 2162-2388. DOI: 10.1109/tnnls.2021.3070843. adr.: http://dx.doi.org/10.1109/TNNLS.2021.3070843.

[45] Z. Xu, M. J. Cruz, M. Guevara i in., "Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering", w *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '24, Washington, DC, USA: Association for Computing Machinery, lip. 2024. adr.: https://doi.org/10.48550/arXiv.2404.17723.

[46] T. Bui, O. Tran, P. Nguyen i in., *Cross-Data Knowledge Graph Construction for LLM-enabled Educational Question-Answering System: A Case Study at HCMUT*, 2024. arXiv: 2404.09296.

[47] M. M. Hussien, A. N. Melo, A. L. Ballardini, C. S. Maldonado, R. Izquierdo i M. Á. Sotelo, *RAG-based Explainable Prediction of Road Users Behaviors for Automated Driving using Knowledge Graphs and Large Language Models*, maj 2024. arXiv: 2405.00449.

[48] X. Jiang, R. Zhang, Y. Xu i in., *HyKGE: A Hypothesis Knowledge Graph Enhanced Framework for Accurate and Reliable Medical LLMs Responses*, kw. 2024. arXiv: 2312.15883.

[49] T. Bratanic, *Constructing knowledge graphs from text using OpenAI functions*, Dostęp zdalny (03.07.2024): https://bratanic-tomaz.medium.com/constructing-knowledge-graphs-from-text-using-openai-functions-096a6d010c17, 2023.

[50] X. Jiang, R. Zhang, Y. Xu i in., *HyKGE: A Hypothesis Knowledge Graph Enhanced Framework for Accurate and Reliable Medical LLMs Responses*, 2024. arXiv: 2312.15883 [cs.CL]. adr.: https://arxiv.org/abs/2312.15883.

[51] Y. Hu, Z. Lei, Z. Zhang, B. Pan, C. Ling i L. Zhao, *GRAG: Graph Retrieval-Augmented Generation*, 2024. arXiv: 2405.16506 [cs.LG]. adr.: https://arxiv.org/abs/2405.16506.

[52] D. Edge, H. Trinh, N. Cheng i in., *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*, 2024. arXiv: 2404.16130 [cs.CL]. adr.: https://arxiv.org/abs/2404.16130.

[53] J. Larson, *GraphRAG: Unlocking LLM discovery on narrative private data*, Dostęp zdalny (04.07.2024): https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-llm-discovery-on-narrative-private-data/, 2024.

[54] Y. Wang, N. Lipka, R. A. Rossi, A. Siu, R. Zhang i T. Derr, *Knowledge Graph Prompting for Multi-Document Question Answering*, 2023. arXiv: 2308.11730 [cs.CL]. adr.: https://arxiv.org/abs/2308.11730.

[55] Z. Xu, M. J. Cruz, M. Guevara i in., "Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering", w *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR 2024, t. 33, ACM, lip. 2024, s. 2905–2909. DOI: 10.1145/3626772.3661370. adr.: http://dx.doi.org/10.1145/3626772.3661370.

[56] J. Wu, J. Zhu i Y. Qi, *Medical Graph RAG: Towards Safe Medical Large Language Model via Graph Retrieval-Augmented Generation*, 2024. arXiv: 2408.04187 [cs.CV]. adr.: https://arxiv.org/abs/2408.04187.

[57] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie i C. D. Manning, *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval*, 2024. arXiv: 2401.18059 [cs.CL]. adr.: https://arxiv.org/abs/2401.18059.

[58] R. Y. Pang, A. Parrish, N. Joshi i in., *QuALITY: Question Answering with Long Input Texts, Yes!*, 2022. arXiv: 2112.08608 [cs.CL]. adr.: https://arxiv.org/abs/2112.08608.

[59] T. Kočiský, J. Schwarz, P. Blunsom i in., *The NarrativeQA Reading Comprehension Challenge*, 2017. arXiv: 1712.07040 [cs.CL]. adr.: https://arxiv.org/abs/1712.07040.

[60] R. DiYanni, *Literature: Reading Fiction, Poetry, Drama, and the Essay*. McGraw-Hill, 1990, ISBN: 9780075571124. adr.: https://books.google.pl/books?id=fXOgtmDNuD4C.

[61] Ragas, *Ragas (0.1.10)*, Dostęp zdalny (03.07.2024): https://docs.ragas.io/en/latest/concepts/index.html, 2024.

[62] P. Gutenberg, *Project Gutenberg - Free eBooks*, Dostęp zdalny (13.08.2024): https://www.gutenberg.org, 2024.

[63] R. Y. Pang, A. Parrish, N. Joshi i in., *QuALITY: Question Answering with Long Input Texts, Yes! - Leaderboard*, 2023. adr.: https://nyu-mll.github.io/quality/.

[64] H. Sienkiewicz, *Ogniem i mieczem*, Polish. Warszawa, Poland: Bellona, 2018, ISBN: 978-8311153677.

Spis rysunków

2.1 Architektura transformera. Ilustracja za [9]. 10
2.2 *Few-shot prompting* z podaniem przykładowych pytań 14
2.3 *In-context learning* - przykład odpowiedzi na pytanie o fakty, które zaszły po zamknięciu zbioru treningowego dla modelu (zrzut ekranu z [1]) 15
2.4 Schemat frameworka RAG (źródło: [19]) 16
2.5 Hierarchia systemów RAG (źródło: [28]) 18
2.6 Przykładowy graf wiedzy (źródło: [41]) 22
2.7 Schemat budowy grafu (za [57]) 29
2.8 Schemat wyszukiwań z użyciem trawersowania drzewa (*Tree Traversal*) i metody powalonego drzewa (*Collapsed Tree*) (za [57]) 30
3.1 Schemat podziału informacji 33
3.2 Schemat tworzenia węzłów wyższego poziomu 36
4.1 Porównanie poprawności odpowiedzi pomiędzy datasetami (wykres) 44
4.2 Porównanie podobieństwa semantycznego odpowiedzi pomiędzy datasetami (wykres) 45
4.3 Wpływ liczby wykorzystanych rezultatów na poprawność odpowiedzi - QuALITY 46
4.4 Wpływ liczby wykorzystanych rezultatów na poprawność odpowiedzi - QuALITY (trudne) 47
4.5 Wpływ liczby wykorzystanych rezultatów na poprawność odpowiedzi - NarrativeQA 47
4.6 Wpływ liczby tokenów na poprawność odpowiedzi - QuALITY 48
4.7 Wpływ liczby tokenów na poprawność odpowiedzi - QuALITY (trudne) 49
4.8 Wpływ liczby tokenów na poprawność odpowiedzi - NarrativeQA 49
4.9 Dataset QuALITY - poprawność odpowiedzi dla pojedynczych aspektów i z ich wyłączeniem 51
4.10 Dataset QuALITY (tylko trudne) - poprawność odpowiedzi dla pojedynczych aspektów i z ich wyłączeniem 52
4.11 Dataset NarrativeQA - poprawność odpowiedzi dla pojedynczych aspektów i z ich wyłączeniem 53
4.12 Porównanie poprawności odpowiedzi w zależności od ilości węzłów uszczegóławiających 54
4.13 Porównanie poprawności odpowiedzi w zależności od długości fragmentów i streszczeń 57

Spis tabel

3.1 Porównanie wykorzystanych modeli 34
4.1 Porównanie poprawności odpowiedzi pomiędzy datasetami 44
4.2 Porównanie podobieństwa semantycznego odpowiedzi pomiędzy datasetami 45