



POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Instytut Informatyki

Praca dyplomowa inżynierska

**INTELIGENTNY SYSTEM UDZIELANIA ODPOWIEDZI NA
PYTANIA WZBOGACONY O KONTEKST WSKAZANY W
WYSZUKANYCH DOKUMENTACH**

Dawid Adam Plaskowski, 139758

Mateusz Politycki, 145263

Marcin Korcz, 144446

Alex Terentowicz, 145419

Promotor

prof. dr hab. inż. Jerzy Stefanowski

POZNAŃ 2023

Spis treści

1 Wprowadzenie	1
1.1 Motywacje do podjęcia tematyki rozprawy	1
1.2 Systemy udzielające odpowiedzi na zadane pytania	2
2 Cel i zakres pracy	4
2.1 Szczegółowy opis zadań pracy	4
2.1.1 Biblioteka	4
2.1.2 Środowisko do testowania promptingu	4
2.1.3 OpenSystem	5
2.1.4 Eksperymenty	5
2.2 Podział zadań w zespole autorów	5
3 Transformer	7
3.1 Przetwarzanie wstępne	8
3.2 Mechanizm samouwagi	8
3.3 Blok enkodera	9
3.4 Blok dekodera	9
3.5 Pretrenowane modele	10
3.6 Warianty architektury	10
4 Wyszukiwanie informacji	12
4.1 Wyszukiwanie leksykalne	12
4.1.1 Okapi BM25	12
4.2 Wyszukiwanie semantyczne	13
4.2.1 SentenceBERT	13
4.2.2 Wyszukiwanie podobieństw w przestrzeni wektorowej	14
4.2.3 Indeksowanie przestrzeni wektorowej	15
FAISS	15
ANNOY	17
4.2.4 Miary odległości	19
Odległość Euklidesowa	19
Podobieństwo kosinusowe	19
5 Modele języka	20
5.1 GPT	20
5.1.1 GPT-3	20
5.1.2 GPT-J	21
5.2 BLOOM 176B	21

5.3	Cohere xlarge	22
6	In-context learning — uczenie w kontekście	24
6.1	Prompting	24
6.1.1	Prompting dynamiczny	25
6.1.2	Chain of thought wnioskowania	25
6.1.3	Self-ask	26
6.1.4	ReAct	26
7	Zbiory dokumentów i pytań	28
7.1	BERTopic	28
7.2	Benchmark	29
7.2.1	Miary oceny	29
Trafność	29	
Dokładne dopasowanie	29	
Pokrycie dokładnego dopasowania	29	
Miara F1	30	
BLEU	30	
7.3	OpenBookQA	30
7.3.1	Analiza dokumentów	31
7.3.2	Analiza pytań	31
7.4	HotpotQA	33
7.4.1	Analiza dokumentów	34
7.4.2	Analiza pytań	34
7.5	Wikipedia	35
8	Proponowany system	36
8.1	Metodyka pracy	36
8.2	Struktura projektu	36
8.3	Architektura	41
8.4	Implementacja	44
8.4.1	Moduł information_retrievers	45
BM25Retriever	45	
FaissRetriever	46	
AnnoyRetriever	46	
WikiAPI	47	
GoogleCustomSearch	47	
8.4.2	Moduł language_models	47
CohereAPI	48	
BLOOM176bAPI	48	
GPTj6bAPI	49	
8.4.3	Data_loaders	49
8.4.4	Jinja2	49
JINJALoader	49	
8.4.5	promptTemplate	49
8.4.6	Klasa PIQARD	50
8.4.7	ChainTrace	50

8.4.8	Rozszerzenia biblioteki	51
	Prompting dynamiczny	51
	Strategia SelfAware	51
	Strategia <i>ReAct</i>	52
8.4.9	Obsługa konfiguracji yaml	53
8.4.10	piqard_api	53
8.4.11	piqard_app	54
8.5	Obsługa błędów	56
8.6	Testy	59
9	Eksperymenty	61
9.1	Ocena wpływu liczby dokumentów podanych jako kontekst dla pytania testowego	62
	9.1.1 Opis konfiguracji systemu	62
	9.1.2 Wyniki	63
	BLOOM	63
	Cohere xlarge	65
	9.1.3 Wnioski szerszej analizy wyników	66
9.2	Ocena wpływu liczby podanych przykładów rozwiązania zadania w promptcie	67
	9.2.1 Opis konfiguracji systemu	67
	9.2.2 Wyniki	68
	OpenBookQA	68
	HotpotQA	70
	9.2.3 Wnioski szerszej analizy wyników	73
9.3	Ocena działania strategii promptingu nazwanej łańcuchem wnioskowania	74
	9.3.1 Opis konfiguracji systemu	74
	9.3.2 Wyniki	74
	OpenBookQA	74
	HotpotQA	76
	9.3.3 Wnioski szerszej analizy wyników	80
9.4	Ocena działania strategii nazwanej dynamicznym promptingiem	80
	9.4.1 Opis konfiguracji systemu	80
	9.4.2 Wyniki	80
	OpenbookQA	81
	9.4.3 Wnioski szerszej analizy wyników	82
9.5	Ocena wpływu kolejności występowania przykładów w promptcie	83
	9.5.1 Opis konfiguracji systemu	83
	9.5.2 Wyniki	84
	OpenBookQA	84
	9.5.3 Wnioski szerszej analizy wyników	85
9.6	Ogólne wnioski przeprowadzonych eksperymentów	86
10	Przykłady użycia	88
10.1	Użycie biblioteki	88
10.2	Użycie systemu	93
11	Podsumowanie	104
11.1	Forma przeprowadzonych eksperymentów	105

11.2 Wybrane modele języka	105
11.2.1 Ograniczenia związane z oknem kontekstowym modeli języka	106
11.2.2 Ograniczenia związane z korzystaniem z API	106
11.3 Strategie promptingu	106
11.4 Wyszukiwanie informacji	107
11.5 Dalsze kierunki rozwoju	107
Literatura	108
A Prompty wykorzystane w pracy	112

Rozdział 1

Wprowadzenie

1.1 Motywacje do podjęcia tematyki rozprawy

Systemy dialogowe, tj. systemy komputerowe przeznaczone do rozmowy z człowiekiem w formie możliwe zbliżonej do konwersacji w języku naturalnym, są ważnym elementem historii informatyki. Ich inspiracje można odnaleźć już w teście odróżnienia konwersacji z człowiekiem i maszyną zaproponowanym przez Alana Turinga, jako tzw. test inteligencji maszyn. W przypadku sztucznej inteligencji można wskazać na rozwój systemu ELIZA [Wei66], jako terapeuty rogeriańskiego opartego na regułach, a później wprowadzanie komercyjnych asystentów głosowych, takich jak np. Siri lub Alexa, aż po bardzo złożony system Watson firmy IBM, np. przewyższający ludzi w grze Jeopardy!. Współczesne systemy dialogowe, które ostatnio zyskały na popularności, to przede wszystkim ChatGPT stworzony przez OpenAI [SZK⁺22], LaMDA będąca propozycją filmy Google [TDFH⁺22], Galactica utrzymywana przez inżynierów Meta [TKC⁺22], lub Sparrow — rozwiązanie firmy DeepMind [GMT⁺22]. Wszystkie one są oparte na tzw. dużych modelach języka (ang. Large Language Models).

Duże modele języka, wykorzystujące architektury sztucznych sieci neuronowych, stanowią znaczący postęp w zakresie przetwarzania języka naturalnego. Umożliwiają to ich imponujące możliwości generatywne, czyli probabilistyczne wytwarzanie sekwencji słów oraz zdolności przechowywania obszernych informacji w swoich parametrach jako rezultat *pretrenowania* na ogromnych korpusach tekstowych. W ostatnich latach stały się one jednym z dominujących rozwiązań w zadaniach wymagających zrozumienia języka naturalnego, m.in. wykrywanie encji nazwanych (ang. Named Entity Recognition) lub maszynowego czytania ze zrozumieniem (ang. machine reading comprehension).

Pomimo obiecujących wyników osiąganych na zadaniach przetwarzania języka naturalnego wraz ze skalowaniem liczby parametrów dużych modeli języka, ich ograniczeniem jest zjawisko tzw. *halucynacji wiedzy*, czyli generowanie wiarygodnie wyglądających stwierdzeń niezgodnych ze stanem faktycznym [SPC⁺21]. Naturalnym rozwiązaniem w uczeniu maszynowym byłoby dostrajanie modelu, ale podejścia związane z tą praktyką mogą być trudne do wdrożenia ze względu na podatność dużych modeli języka na *katastrofalne zapominanie* (ang. catastrophic forgetting), czyli tendencję do nagłego zapominania poprzednio wyuczonych informacji na skutek dostarczania nowych danych. Z drugiej strony, metody uczenia poprzez wzmacnianie są bardziej odporne na tę wadę [KEKD22]. Pomimo tego, nawet w przypadku wykorzystania najlepszych obecnie technik (ang. state of the art) łączących uczenie nadzorowane z uczeniem przez wzmacnianie w modelu ChatGPT, problem halucynacji wiedzy nie został całkowicie rozwiązany, m.in. z powodu braku zweryfikowanych źródeł wiedzy w zbiorze treningowym [SZK⁺22]. Ponadto proces dostrajania sztucznych sieci neuronowych wymaga znacznych nakładów obliczeniowych, co może być szczególnie problematyczne w

przypadku ciągle zmieniającej się wiedzy o świecie. Wobec tych ograniczeń uzasadnione staje się poszukiwanie alternatywnych sposobów na poprawienie dokładności modeli języka bez konieczności ich douczania do specyfiki kolejnego zadania.

1.2 Systemy udzielające odpowiedzi na zadane pytania

Zadanie automatycznego odpowiadania na pytania (ang. Question Answering) jest uproszczoną formą dialogu, w którym system próbuje udzielić poprawnej odpowiedzi na podane pytanie, nie przechowując i nie adaptując się do historii interakcji z użytkownikiem. Rozróżnia się dwa podstawowe warianty tego zagadnienia, według których można sklasyfikować istniejące rozwiązania [ZLW⁺21].

Pierwszy z nich, czyli *maszynowe czytanie ze zrozumieniem* (ang. Machine Reading Comprehension) zakłada, iż system udziela odpowiedzi na pytanie dotyczące podanego przez użytkownika fragmentu tekstu. Zadanie to można porównać do wersji znanej z egzaminów językowych, gdzie testowana osoba najpierw analizuje dany paragraf, a następnie odpowiada na dotyczącego pytanie. Niestety taka forma ogranicza zdolności systemu jedynie do zagadnień związanych z konkretnym tekstem, który został uprzednio przygotowany przez użytkownika.

W ramach rozszerzenia powyżej opisanego wariantu rozważa się problem *odpowiedzi na pytania dotyczące wielu dziedzin* (ang. Open-Domain Question Answering). Zakłada on brak podanego przez użytkownika fragmentu tekstu, co pozwala na większą swobodę w wyborze zagadnień i możliwość dowolnego odpytania systemu zarówno o fakty ogólne, jak i szczegółowo określone tematy. Duże modele języka naturalnego są zdolne zapamiętać w swoich parametrach informacje wyuczone z ogromnych korpusów tekstowych, w związku z czym istnieje możliwość ich bezpośredniego zastosowania [RRS20]. Takie podejście ma jednak swoje ograniczenia spowodowane zarówno zastosowaną wielkością głębokich sieci neuronowych, jak i potrzebą aktualizacji parametrów w związku z dynamicznie zmieniającymi się informacjami na świecie. Pozytywny wpływ na jakość generowanych wyników może mieć wzbogacenie modeli języka o moduł *wyszukiwania informacji* [PLP⁺20]. Odpowiada on za odnajdywanie, w lokalnych bądź zdalnych repozytoriach, dokumentów zawierających informacje relevantne w stosunku do zadанego pytania. Są one następnie traktowane jako jego kontekst i stają się dodatkową przesłanką dla systemu w trakcie udzielania odpowiedzi.

Jednym ze znanych schematów wykorzystującym połączenie modelu języka naturalnego z wyszukiwaniem informacji jest *Retriever-Reader* [CFWB17], który dekomponuje problem na dwa etapy. Pierwszy z nich, czyli wyszukanie kontekstu, ma na celu odnalezienie informacji w zewnętrznych repozytoriach dokumentów. Wykorzystuje się tutaj zarówno klasyczne techniki wyszukiwania leksykalnego np. za pomocą algorytmu Okapi BM25, jak i bardziej zaawansowane wyszukiwanie semantyczne oparte na wektoryzacji dokumentów przez głębokie sieci neuronowe. W drugim etapie wyselekcjonowany kontekst jest przetwarzany przez modele języka naturalnego, aby wyszczególnić konkretny fragment, który zawiera odpowiedź. W tym celu stosuje się głównie rozwiązania oparte na modelu BERT [DCLT18], który po dotrenowaniu okazuje się osiągać wysoką trafność odpowiedzi [YXL⁺19a].

Powyżej opisany schemat reprezentuje podejście tzw. *ekstraktywnego udzielania odpowiedzi* (ang. Extractive Question Answering), który charakteryzuje się zaznaczaniem wybranego fragmentu w tekście, jako ostatecznego rozwiązania. Niestety głównym ograniczeniem tego podejścia jest założenie, że w wyszukanym kontekście znajduje się bezpośrednią odpowiedź na pytanie. Dodatkowo, połączenie odległych od siebie informacji w dokumencie jest możliwe jedynie kosztem jakości ostatecznie prezentowanego wyniku.

Alternatywą do powyższego podejścia jest rozwiązywanie znane jako *generatywne udzielanie odpowiedzi* (ang. Generative Question Answering), które zamiast wybierać fragment kontekstu, tworzy własną odpowiedź, bazując na wyuczonych parametrach. W tym celu stosuje się generatywne modele języka naturalnego, których wiodącym przykładem jest GPT-3 [BMR⁺20], a ich działanie opiera się na przewidywaniu kolejnego tokenu w sekwencji.

Tworzy się także złożone podejścia wykorzystujące generatywne modele języka wraz z wyszukiwaniem informacji. W taki sposób powstał schemat nazwany *Retriever-Generator*, który podobnie jak Retriever-Reader, składa się z dwóch etapów i w pierwszym z nich wyszukiwany jest odpowiedni dla zadanego pytania kontekst. Różnica występuje w kolejnej fazie, gdzie zamiast zaznaczania fragmentu, model języka generuje odpowiedź, korzystając z własnych parametrów oraz dodatkowych informacji odnalezionych we fragmencie tekstu. Zalety zastosowania modeli generatywnych widoczne są nie tylko w tworzeniu unikalnej odpowiedzi na podstawie wnioskowania po wielu fragmentach kontekstu, ale także w możliwości wpływu na ich działanie za pomocą metody uczenia w kontekście opartej na przygotowanych wzorcach wzbudzających model, zwanych dalej *promptami*.

Uczenie w kontekście polega na dostosowaniu wygenerowanej przez model języka odpowiedzi poprzez przekazanie mu wzorca zawierającego instrukcję dotyczącą zadania wraz z przykładami jego rozwiązania. Dzięki zastosowaniu tej metody możliwe jest m.in., aby udzielona odpowiedź była *łańcuchem wnioskowania* w postaci listy kolejnych kroków prowadzących do danego rozwiązania (ang. chain of thought) [WWS⁺22b]. Metoda ta pozwala również wykorzystać sekwencyjne przetwarzanie modeli generatywnych poprzez umieszczenie przed docelowym pytaniem kilku przykładów rozwiązania manualnie napisanych bądź dynamicznie dobranych z dostępnego zbioru zadań (ang. few-shot learning).

W oparciu o możliwości modeli generatywnych w połączeniu z zastosowaniem metody uczenia w kontekście, a także wyszukiwania informacji w zewnętrznych źródłach powstało wiele innowacyjnych rozwiązań. Przykładem mogą być strategie promptingu m.in. *ReAct* [YZY⁺22], który traktując model jako agenta sekwencyjnie tworzy łańcuch akcji, np. żądanie wyszukania informacji, na które następują konkretne reakcje prowadzące system do ostatecznej odpowiedzi. Inną strategią jest *self-ask* [PZM⁺22] dekomponujący złożone pytania na kilka prostszych. Generując odpowiedzi na kolejne z nich, zyskuje częściowe informacje, które pozwalają rozwiązać oryginalny problem.

Powyższe przykłady ukazują potencjał i szeroką gamę możliwych rozwiązań problemu odpowiedzi na zadane pytanie, które wraz z rozwojem nowych koncepcji oraz metod opartych zarówno na technologiach komercyjnych, jak i open-source, osiągają coraz lepsze wyniki.

Rozdział 2

Cel i zakres pracy

Kierując się motywacjami przedstawionymi w poprzednim rozdziale, głównym celem naszej rozprawy jest stworzenie systemu odpowiadającego na pytania z wielu dziedzin w języku angielskim, który wykorzystuje modele neuronowe języka rozszerzone przez wyszukany kontekst pytania.

Chcemy zwrócić szczególną uwagę na kwestie związane z kosztami obliczeniowymi oraz trudnościami z wczytaniem wag modelu na sprzęcie konsumenckim. W celu uniknięcia tych ograniczeń wykorzystamy modele z udostępnionym API, a konkretniej BLOOM 176B oraz Cohere xlarge. Dodatkowo użyjemy metod wpływających na zachowanie modelu w oparciu o uczenie w kontekście, ale niewymagających jego dalszego trenowania np. zastosowanie strategii łańcucha wnioskowania.

Kolejnym ważnym celem naszej rozprawy jest przeprowadzenie szerokich eksperymentów sprawdzających wpływ wybranych strategii promptingu na jakość generowanych odpowiedzi przez modele BLOOM 176B i Cohere xlarge. Ocenę eksperymentalną wykonamy z wykorzystaniem tzw. benchmarków o nazwach OpenBookQA oraz HotpotQA. Pierwszy z nich zawiera pytania wielokrotnego wyboru dotyczące faktów naukowych, natomiast kolejny składa się z pytań otwartych zebranych na podstawie zapytań użytkowników do wyszukiwarki internetowej.

Praktycznym rezultatem naszej pracy będzie biblioteka open-source w języku Python wraz z aplikacją internetową, która udostępnia interfejs użytkownika i umożliwia uzyskanie odpowiedzi na zadane pytanie w języku angielskim wraz ze szczegółami.

2.1 Szczegółowy opis zadań pracy

Realizacja powyżej wymienionych głównych celów pracy związana jest ze sformułowanie bardziej szczegółowych zadań, które omawiamy poniżej.

2.1.1 Biblioteka

Biblioteka w języku Python 3 umożliwi użytkownikowi zbudowanie systemu z gotowych komponentów oraz testowanie nowych koncepcji wykorzystania modeli języka naturalnego BLOOM 176B i Cohere xlarge w połączeniu z wyszukiwaniem kontekstu w zewnętrznych repozytoriach dokumentów metodą leksykalną Okapi BM25, a także metodami semantycznymi FAISS i ANNOY. Dodatkowo udostępnione zostaną wzorce promptingu, które umożliwiają natychmiastowe wykorzystanie metody uczenia w kontekście m.in., ReAct oraz moduł pozwalający wczytać własne prompty.

2.1.2 Środowisko do testowania promptingu

Proces poszukiwania odpowiednich promptów może być czasochlonny i wymagać wielu iteracji redagowania instrukcji, ponieważ modele często generują nieoczekiwane wyniki. W celu umożliwienia

wienia bardziej efektywnego prototypowania oraz zapobiegania takim niepożądanym zachowaniom chcemy zapewnić użytkownikowi środowisko pozwalające na bezpośrednie pisanie nowych promptów, wybieranie modułu wyszukiwania informacji i obserwowanie ich wpływu na generowane odpowiedzi.

2.1.3 OpenSystem

Skonfigurowany system, nazwany przez nas PIQARD (Prompted Intelligent Question Answering with Retrieval of Documents), umożliwi uzyskanie odpowiedzi na zadane pytanie w języku angielskim wraz z podanymi informacjami dotyczącymi częściowego sposobu wytworzenia rozwiązania. Mogą to być zarówno kolejne kroki danej strategii promptingu, jak i wyszukany w dokumentach kontekst. Interakcje użytkownika z systemem zapewni interfejs w formie aplikacji internetowej zbudowanej w języku Javascript z frameworkm React.js.

2.1.4 Eksperymenty

Przetestowanie zachowania wybranych modeli języka naturalnego BLOOM 176B oraz Cohere xlarge na wyselekcjonowanych benchmarkach OpenBookQA oraz HotpotQA, wzbudzając je poszczególnymi strategiami promptingu m.in.:

- podanie instrukcji wraz z przykładami rozwiązania podobnych zagadnień (few-shot prompting)
- dodanie wyszukanej informacji jako kontekst do promptingu
- wymuszenie łańcucha wnioskowania
- dynamiczne tworzenie przykładów dla strategii few-shot prompting

pozwoli na ocenę trafności oraz jakości generowanych odpowiedzi według wybranych miar oceny. Sprawdzą one, która z zaproponowanych strategii działa lepiej w zadaniu udzielenia odpowiedzi na pytania wielokrotnego wyboru oraz odpowiedzi na pytania otwarte.

2.2 Podział zadań w zespole autorów

Rozwój pracy inżynierskiej oraz implementacja proponowanego systemu została podzielona na czterech członków zespołu według poniżej opisanych zadań:

- **Dawid Plaskowski**
 - Przegląd literatury dotyczącej zagadnienia odpowiedzi na pytania oraz modeli języka naturalnego BLOOM 176B i rodziny GPT.
 - Analiza artykułów przedstawiających strategie promptingu m.in. łańcuch wnioskowania, Self-ask, czy ReAct. Analiza dostępnych rozwiązań łączących wyszukiwanie informacji z modelami języka, a także sposobów modelowania tematycznego
 - Implementacja strategii łańcucha wnioskowania, SelfAware i ReAct
 - Stworzenie wykorzystywanych promptów (ang. prompt engineering)

• **Marcin Korcz**

- Przegląd literatury dotyczącej wyszukiwania leksykalnego oraz semantycznego, w szczególności analiza algorytmów BM25 oraz ANNOY
- Integracja algorytmów BM25 oraz ANNOY z systemem
- Stworzenie testów jednostkowych
- Modelowanie tematyczne wybranych zbiorów danych oraz benchmarków

• **Mateusz Politycki**

- Przegląd literatury dotyczącej modeli języka naturalnego BLOOM 176B oraz rodziny GPT
- Analiza oraz wybranie benchmarków do przeprowadzenia eksperymentów
- Przeprowadzenie eksperymentów ze strategiami promptingu m.in. dynamiczny prompting, łańcuch wnioskowania na modelach BLOOM 176B oraz Cohere xlarge
- Implementacja architektury systemu wraz ze stworzeniem koncepcji modułów oraz przetwarzania konfiguracji
- Stworzenie interfejsu użytkownika w postaci aplikacji internetowej

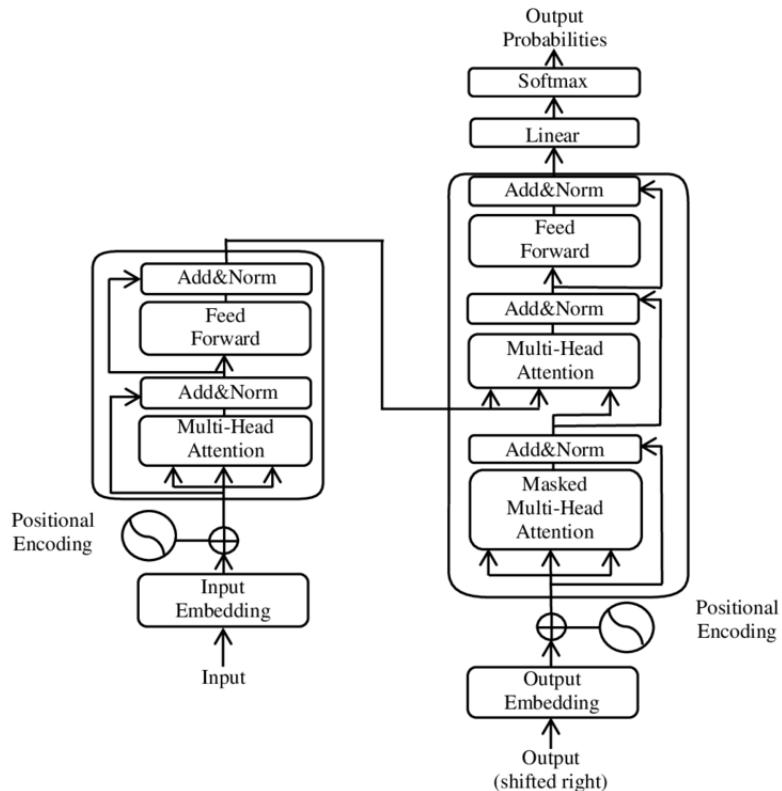
• **Alex Terentowicz**

- Przegląd literatury dotyczącej wyszukiwania semantycznego, w szczególności analiza algorytmu FAISS, a także sprawdzenie dostępnych modeli wektoryzacji
- Przeprowadzenie wektoryzacji wybranych zbiorów danych wraz z odpowiednimi benchmarkami służącymi do testowania
- Integracja algorytmu FAISS z systemem
- Stworzenie wykorzystywanych promptów (ang. prompt engineering)

Rozdział 3

Transformer

Transformer [VSP⁺17] to architektura sieci neuronowej przeznaczona do modelowania danych sekwencyjnych, która zdobyła początkowo zastosowanie w przetwarzaniu języka naturalnego, ale zaczyna być szeroko stosowana w innych dziedzinach uczenia głębokiego. Swój sukces zawdzięcza wykorzystaniu metod przetwarzania sekwencji bez użycia elementów sieci rekurencyjnych, co pozwala na uniknięcie konieczności sekwencyjnego przetwarzania danych. Przełomowym rozwiązaniem zastosowanym w architekturze okazał się mechanizm *samouwagi* (ang. self-attention) umożliwiający równolegle obliczenia i co za tym idzie wydajniejsze wykorzystanie zasobów obliczeniowych w trakcie uczenia. Oryginalna architektura składała się z enkodera i dekodera przedstawionego na rysunku 3.1.



RYSUNEK 3.1: Architektura transformer, źródło ilustracji: [Jia19]

3.1 Przetwarzanie wstępne

W przypadku danych tekstowych kluczowym krokiem w procesie przetwarzania jest tokenizacja, czyli redukcja tekstu do atomowej postaci słów, które występują w ustalonym słowniku. Następnie, dla każdego z tokenów, wyliczane są ich wektorowe reprezentacje tzw. *zanurzenia* (ang. embeddings). Warto zauważyć, że mechanizm samouwagi jest niezmienniczy względem permutacji (ang. permutation invariant) - traktuje słowa jako zbiór, tracąc przy tym ich kolejność, dlatego konieczne jest uwzględnienie informacji o pozycji każdego z tokenów w sekwencji. W celu rozwiązania tego problemu, autorzy architektury transformera zdecydowali się zastosować *zanurzenia pozycyjne* (ang. positional encodings), które dodają informację o szyku, opierając się na wybranych wcześniej funkcjach okresowych:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \quad (3.1)$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \quad (3.2)$$

Gdzie d_{model} oznacza wymiarowość wejścia i wyjścia, a pos to pozycja tokenu. Alternatywnie można skorzystać z wyuczonych reprezentacji zanurzeń pozycyjnych. W obu przypadkach są one dodawane do zanurzeń poszczególnych tokenów i razem stanowią sekwencję wejściową do modelu.

3.2 Mechanizm samouwagi

Mechanizm samouwagi pozwala na uwzględnienie relacji poszczególnych tokenów, wzbogacając ich wektory wejściowe o kontekst występowania każdego elementu w sekwencji. Model przetwarzając reprezentację wektorową danego tokenu, wykorzystuje informację tokenów na innych pozycjach, tworząc asocjacje pomiędzy nimi. Jedną z głównych zalet transformera jest możliwość ujęcia zależności i tworzenia złożonych reprezentacji dla dowolnie oddalonych od siebie tokenów. W przeciwieństwie do sieci rekurencyjnych złożoność obliczeniowa w transformerze jest stała, niezależnie od odległości pomiędzy tokenami. Korzysta się do tego zadania z trzech macierzy przechowujących parametry optymalizowane w trakcie uczenia: macierz zapytań W^Q , kluczy W^K i wartości W^V . Dla każdego elementu na pozycji i wektory klucza, wartości i zapytania są wyliczane jako transformacje liniowe wejściowego zanurzenia x_i :

$$k_i = W^K x_i \quad (3.3)$$

$$v_i = W^V x_i \quad (3.4)$$

$$q_i = W^Q x_i \quad (3.5)$$

Następnie wyliczane jest znormalizowane podobieństwo kluczy do zapytania:

$$\alpha_j = softmax\left(\frac{q_i^T k_j}{\sqrt{d_k}}\right) \quad (3.6)$$

Każdy z wektorów określający klucz jest sparowany z wektorem wartości. Im większa zgodność pomiędzy zapytaniem a kluczem, tym większy wpływ wektor wartości będzie miał na dany element. Dodatkowo wzór 3.6 skaluje się wynik przez wartość $\sqrt{d_k}$ za względu na rosnącą wariancję iloczynu skalarnego wraz ze zwiększeniem rozmiaru wektora klucza d_k . Ostateczną reprezentację wzbogaconą o kontekst, jako wynik mechanizmu samouwagi dla danego wektora, uzyskujemy licząc kombinację liniową wektorów wartości i ich podobieństw:

$$h_i = \sum_{j=1}^n \alpha_j v_j \quad (3.7)$$

Wektory w mechanizmie samouwagi można połączyć w macierze Q, K, V i na nich dokonywać wszelkich operacji, co sprzyja ich zrównolegleniu. Funkcja samouwagi przyjmuje wtedy wzór:

$$\text{Samouwaga}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3.8)$$

W tekście istnieje wiele różnych relacji między słowami, dlatego stosuje się kilka bloków mechanizmu samouwagi o różnych macierzach parametrów W^Q, W^K, W^V , aby te zależności uwzględnić w różnorodnych reprezentacjach. Wielogłowicowy mechanizm uwagi składa się z h równolegle przetwarzanych bloków samouwagi, tzw. głowic (ang. multi-head attention). Wyjściowe wektory z każdego bloku są łączone w jeden wektor rzutowany za pomocą macierzy wyjściowej W^O .

$$\text{głowica}_i = \text{Samouwaga}(QW_i^Q, KW_i^K, VW_i^V) \quad (3.9)$$

$$\text{WielogłowicowaSamouwaga}(Q, K, V) = \text{Concat}(\text{głowica}_1, \dots, \text{głowica}_h)W^O \quad (3.10)$$

3.3 Blok enkodera

W architekturze transformera występują *połączenia rezydualne* (ang. residual connections), w których łączą się wynik bloku wielogłowicowej samouwagi z sekwencją wejściową. Oprócz polepszenia zbieżności procesu uczenia się sieci przez poprawę przepływu gradientu [HZRS15], połączenia rezydualne odgrywają również ważną rolę w przesyłaniu informacji o pozycji tokenów do kolejnych warstw. Kolejnym wyborem architektury jest normalizacja warstwy (ang. layer normalization), po której występują dwie warstwy gęste z funkcją aktywacji ReLU po pierwszej z nich. Następne jest ponowne połączenie rezydualne tym razem łączące wyjście sieci w pełni połączonej z jej wejściem oraz normalizacja warstwy zwracająca już ostateczny wynik. Blok składający się z wielogłowicowej warstwy samouwagi, połączeń rezydualnych, normalizacji warstw i sieci w pełni połączonej stanowi jeden blok enkodera. Zwykle bloków jest wiele, a w oryginalnym pomyśle autorów użyto stosu sześciu bloków enkodera.

3.4 Blok dekodera

Blok dekodera jest podobny do bloku enkodera, jednak zawiera dodatkową warstwę uwagi enkoder-dekoder, która przetwarza wynik działania stoso enkoderów. Ta warstwa jest kluczowa do połączenia wzbogaconej reprezentacji wejściowej z generowaniem sekwencji wyjściowej. W celu uniknięcia problemu dostępu dekodera do nieprzetworzonych jeszcze tokenów w trakcie uczenia dla danej sekwencji wejściowej, w warstwie samouwagi w bloku dekodera, zastosowano metodę maskowania przyszłych tokenów. Jest to osiągane poprzez nadanie wartości $-\infty$ elementom macierzy w górnej przekątnej, dzięki czemu są one wyzerowane po warstwie normalizującej softmax. Przykładem pierwotnego zastosowania transformera jest tłumaczenie maszynowe, gdzie reprezentacja całej sentencji w języku źródłowym jest uzyskiwana przez stos enkoderów, a generowanie kolejnych słów w języku docelowym jest ograniczone do poprzednio wygenerowanych słów.

Po przetwarzaniu danych przez stos dekoderów (liczba bloków jest taka sama jak liczba bloków enkodera), wyjściowa reprezentacja przechodzi przez warstwę liniową, która rzutuje wynik na wektor o wymiarze odpowiadającym rozmiarowi użytego słownika. Ten wektor reprezentuje rozkład prawdopodobieństwa wystąpienia każdego słowa jako kolejnego w sekwencji wyjściowej po normalizowaniu przez funkcję softmax. W trakcie uczenia wybierane jest słowo metodą zachłanną, a następnie jest ono dodawane do sekwencji wejściowej. Cały proces jest powtarzany dla kolejnego tokenu, aż do osiągnięcia określonego warunku końca generowania sekwencji.

Warto zaznaczyć, że złożoność obliczeniowa warstwy samouwagi zależy od kwadratu długości sekwencji wejściowej, dlatego ogranicza się długość przetwarzanej sekwencji — w artykule wprowadzającą tę architekturę wynosi ona 512. [Ala18]

3.5 Pretrenowane modele

Pretrenowany model można zdefiniować jako zapisaną architekturę modelu uczenia maszynowego wraz z wagami wyuczonymi wcześniej na zbiorze treningowym. Wykorzystuje się je zarówno jako bezpośrednie rozwiązywanie problemu bez konieczności wprowadzenia dodatkowych modyfikacji, ale także jako punkt startowy w metodyce zwanej *transferem wiedzy* bądź *transferem początkowych wartości wag* (ang. transfer learning). Jej głównym założeniem jest przeniesienie zdobytych w trakcie szkolenia informacji na inne, lecz podobne zadanie. Zamiast zaczynać trening od początku wystarczy jedynie dotrenać model na danych dotyczących nowego problemu korzystając z wcześniej wyuczonych wzorców. Pozwala to oszczęścić zarówno potrzebne zasoby obliczeniowe, jak i skrócić wymagany czas uczenia.

3.6 Warianty architektury

Transformer ma kilka wariantów, w tym T5 i BART, które trzymają się oryginalnej koncepcji połączenia enkodera z dekoderem w zadaniach przekształcenia jednej sekwencji na inną (tzw. seq2seq), ale rozszerzają ją o techniki maskowanego modelowania języka, zasumienia przez usuwanie tokenów lub ich permutacje. Są też modele, takie jak BERT i GPT, które skupiają się tylko na jednym z bloków — enkoderze lub dekoderze.

BERT (skrót z ang. Bidirectional Encoder Representations from Transformers) [DCLT18] jest modelem do tworzenia reprezentacji języka naturalnego poprzez uwzględnienie kontekstu słów w zdaniu w sposób dwukierunkowy — przetwarzając tekst od początku do końca i od końca do początku. Autorzy wybrali algorytm WordPiece do tokenizacji tekstu i wprowadzili specjalne tokeny: CLS służący do klasyfikacji, SEP do rozdzielania dwóch zdań, MASK stosowany do maskowania tokenów. Podczas uczenia modelu w etapie pretreningu, jako sekwencję wejściową podaje się dwa zdania oddzielone separatorem SEP (dodatkowo do sekwencji na początku dołącza się token CLS odpowiadający za reprezentację zdania przeznaczoną do zadań klasyfikacji) i polega się na dwóch zadaniach:

1. *maskowanego modelowania języka* (ang. Masked Language Modeling), w którym losowo zakrywa się tokeny z ustalonym prawdopodobieństwem, a zadaniem jest rekonstrukcja początkowego zdania przez predykcję słów w zamaskowanych miejscach. W [DCLT18] prawdopodobieństwo wynosiło 15%, a wylosowane tokeny w 80% zostały zastąpione tokenem MASK, 10% podmienione losowym słowem i w 10% przypadków nie wprowadzono żadnych zmian.
2. *predykcji kolejnego zdania* (ang. Next Sentence Prediction), pomagającą uchwycić zależności pomiędzy zdaniami poprzez klasyfikację czy zdanie występujące w sekwencji wejściowej po prawej stronie separatora [SEP] następuje po zdaniu z lewej strony. Autorzy, starając się, aby w zadaniu predykcji kolejnego zdania klasy były zbalansowane, zautomatyzowali proces ich dobierania gwarantując podział 50% dla każdej z klas.

Istnieją dwie wersje modelu, *BERT_{base}* złożonego z 12 bloków enkodera i 110 milionami parametrów oraz *BERT_{large}* złożonego z 24 bloków enkodera i 340 milionami parametrów, a wyuczone

zostały na anglojęzycznej Wikipedii (2.5 miliarda słów) oraz BooksCorpus (800 milionów słów). Modele bazujące na enkoderze, takie jak BERT i jego rozszerzenia, stanowią podstawę współczesnych systemów wielozadaniowych w przetwarzaniu języka naturalnego, np. do klasyfikacji tekstu, analizy wydawnictwa lub wykrywania encji nazwanych, co zawdzięczają metodzie pretreningu i transferu wiedzy. W przypadku wyszukiwania informacji, transfer wiedzy pozwala na dostosowanie pretrenowanych modeli przez dostrajanie ich do tworzenia kontekstowych reprezentacji, w których semantycznie podobne zdania są zanurzane blisko siebie. Jest to skutecznie wykorzystywane w module wyszukiwania semantycznego w naszym systemie opisany w rozdziale 4 na stronie 12. BERT jest również stosowany w metodzie ekstrakcji odpowiedzi, w której model ma za zadanie znaleźć fragment tekstu zawierający odpowiedź na zadane pytanie [YXL^{+19b}].

O transformerze opartym o dekoder, stosowanym jako model języka w naszej pracy, napisaliśmy w rozdziale 5 na stronie 20. zatytułowanym “Modele języka”. Wyodrębniliśmy ten wariant architektury, ponieważ stanowi on główny moduł udzielania odpowiedzi na pytania w koncepcji naszego systemu i eksperymentów.

Rozdział 4

Wyszukiwanie informacji

Wyszukiwanie informacji to proces polegający na wyborze w zbiorach danych informacji, która jest tzw. relevantna w stosunku do zapytania. W celu dobrego wyszukania takiej informacji stosuje się różne sposoby pomiaru jej reprezentacji do wyrażania w zapytaniu. Najczęściej są to podobieństwo kosinusowe lub odległość euklidesowa opisane m.in. w książce [MRS08]. Powstało także wiele metod i algorytmów wyszukiwania opartych zarówno na licznosci występujących słów, nazwane wyszukiwaniem leksykalnym np. BM25 [MRS08], jak i ich znaczeniu, określonym jako wyszukiwanie semantyczne np. FAISS [JDJ17].

4.1 Wyszukiwanie leksykalne

Wyszukiwanie leksykalne to metoda wyszukiwania informacji za pomocą słów kluczowych. Indeksuje się w niej dokumenty tworząc ich zastępczą reprezentację wektorową, która może być oparta na zliczaniu częstości występujących w nich słów. Korzystając z tej reprezentacji, można stworzyć ranking najbardziej dopasowanych dokumentów na podstawie wartości wybranej miary podobieństwa. Jest to jedna z podstawowych metod i stanowi ważny element wielu systemów informatycznych. Systemy wyszukiwania leksykalnego są ograniczone tym, że nie rozumieją intencji wyszukującego, ani kontekstu, w jakim informacja jest przedstawiona. Może to stanowić problem, jeżeli w instrukcji wyszukiwawczej użyto innego słownictwa niż zawarte w szukanym dokumencie, nawet jeśli niosą ze sobą tę samą informację.

4.1.1 Okapi BM25

Okapi BM25 (Best Match 25) to często stosowana i efektywna funkcja tworząca ranking dokumentów. Należy do probabilistycznych modeli wyszukiwania informacji i stanowi alternatywę dla prostszych algorytmów, m.in. TF-IDF [SB88]. Korzysta z reprezentacji wektorowej dokumentów w postaci tzw. "worka słów" (ang. Bag of words) i ocenia ich zbiór względem występujących termów danego zapytania. Term w tym kontekście można zdefiniować jako występujące słowo w postaci bazowej bądź jego przetworzonej wstępnie formie np. poprzez lematyzacje. Istnieją różne warianty, jednak jeden z najważniejszych opisów matematycznych funkcji wygląda następująco [Wik22a]:

$$ocena(D, Q) = \sum_{i=1}^n IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \quad (4.1)$$

gdzie:

Q - Zapytanie zawierające q_1, \dots, q_n termów

D - Dokument

$|D|$ - Długość dokumentu

$f(q_i)$ - Liczba wystąpień termów q_i w dokumencie D.

$avgdl$ - Średnia długość dokumentu w zbiorze

k_1, b - Parametry funkcji

$IDF(q_i)$ - odwrotna częstość w dokumentach (ang. inverse document frequency). Wyznacza wagę termów q_i . Istnieją odmiany sposobu wyliczania tego współczynnika, lecz najczęściej ma on następującą postać:

$$IDF(q_i) = \ln \left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1 \right) \quad (4.2)$$

gdzie:

N - liczba wszystkich dokumentów w zbiorze

$n(q_i)$ - liczba dokumentów zawierających term q_i

BM25 można dostrajać do potrzeb wyszukiwania za pomocą parametrów k_1 oraz b . Wartości optymalne zależą od zbioru dokumentów, jednak jak wspomina autor pracy [RZ09], najlepsze wyniki uzyskuję się, gdy parametr b jest w przedziale [0.5, 0.8], a parametr k_1 [1.2, 2].

4.2 Wyszukiwanie semantyczne

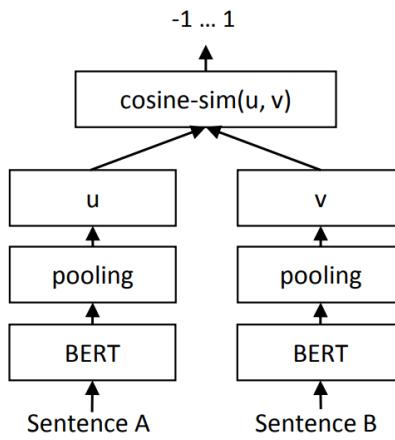
Metoda wyszukiwania semantycznego stanowi alternatywę do dopasowania słów kluczowych lub oszacowania podobieństw termów w wyszukiwaniu leksykalnym. W przeciwieństwie do poprzedniego rozwiązań opartego w głównej mierze na zliczaniu częstości słów lub termów oraz traktowaniu ich jednoznacznie, wyszukiwanie semantyczne polega na przeszukaniu dokumentów pod względem zawartej w nich informacji oraz kontekstowego znaczenia wyrazów. Aktualnie, standardowym podejściem jest połączenie dwóch metod. Najpierw tekst przekształca się do postaci wektorowej (z j. ang. encoding, *enkoduje*) za pomocą modeli głębokich sieci neuronowych opartych na architekturze transformera, a następnie w tak otrzymanym zbiorze wyszukuje się najbliższych sąsiadów, wykorzystując wybraną miarę odległości.

W wyszukiwaniu semantycznym wykorzystuje się blok enkodera, ponieważ pozwala on na otrzymanie liczbowych reprezentacji dokumentów, które przechowują informacje o zawartym w nich kontekście, a także pozwalają określić stopień *podobieństwa znaczeniowego* między nimi. Wysoki stopień *podobieństwa znaczeniowego* porównywanych tekstów określić można tym, że w swojej treści odnoszą się do tych samych obiektów lub koncepcji, w podobnym charakterze. W naszej pracy nie będą nas interesować emocje zawarte w dokumencie, ponieważ poszukujemy faktycznych odpowiedzi na pytania dotyczące świata rzeczywistego. Przy wyborze pretrenowanego modelu do *enkodowania*, najważniejszą cechą jest zdolność do wytworzenia takich *zanurzeń wektorowych* (ang. embeddings), aby teksty powiązane znaczeniowo znalazły się blisko siebie w przestrzeni wektorowej, nawet jeżeli składają się z zupełnie różnego słownictwa.

4.2.1 SentenceBERT

Na przestrzeni lat pojawiło się wiele prac badawczych związanych z dostrajaniem modelów opartych na architekturze BERT do określonych zadań i ulepszania ich pod kątem pozyskiwania wiarygodnych zanurzeń tekstów. W 2019 roku opublikowano wyniki badań nad metodą dostrajania sieci o nazwie Sentence-BERT [RG19] przystosowaną do enkodowania w taki sposób, aby zanurzenia dwóch semantycznie podobnych tekstów znajdowały się blisko siebie w przestrzeni wektorowej. Aby to osiągnąć, w procesie dostrajania zastosowano *sieci syjamskie* (ang. siamese) oraz *trójkowe* (ang. triplets), czyli dwie lub trzy identyczne sieci BERT dostrajane równolegle, każda otrzymując

na wejściu inny tekst, przy czym teksty te stanowią zestaw przykładów pozytywnych lub negatywnych. Przykłady pozytywne to takie, których treść jest semantycznie podobna, a więc dotyczy tych samych tematów i pojęć. Negatywne przykłady to natomiast dokumenty o treściach zupełnie niepowiązanych ze sobą. Zestawem podanym na wejście w trakcie dostrajania może być więc para przykładów pozytywnych, para negatywnych, lub dla sieci trójkowej może być to zestaw składający się z jednego tekstu referencyjnego, jednego tekstu pozytywnego do niego i jednego negatywnego. Wagi dostrajanych tak sieci są tak samo aktualizowane, ponieważ wiąże je wspólną funkcją straty minimalizującą miarę odległości między tekstami podobnymi (pozytywnymi) oraz maksymalizującą dla przykładów przeciwnistwanych (negatywnych). Wybór funkcji zależy od tego, czy w wybranym zbiorze danych oznaczone zostały pary negatywne lub pozytywne.



RYSUNEK 4.1: Proces dostrajania sieci zaproponowaną przez autorów Sentence-BERT [RG19]. Dwie sieci BERT powiązane wspólną funkcją straty aktualizują wagi na podstawie dwóch dokumentów podobnych lub niepowiązanych semantycznie.

Autorzy podkreślili w swojej pracy, że zastosowanie proponowanej metody znacznie zmniejsza koszt obliczeniowy i efektywność wyników dla zadań wymagających semantycznie znaczących *zauważen* tekstu. Przede wszystkim przyspieszony został proces przetwarzania bazy tekstów na zbiór wektorów gotowy do wyszukiwania semantycznego, który twórcy ograniczyli do pojedynczego przetworzenia dokumentu przez model celem otrzymania jednego wektora pozwalającego na obliczenie podobieństwa z wszystkimi innymi tekstami. Poprzednie konkurencyjne metody oparte na technice BERT wymagały wykonywania osobnych obliczeń dla każdej możliwej pary tekstów w bazie. Poprawione zostały też wyniki modelu dla miliardów rang Spearmana na zbiorze STS (Semantic Textual Similarity) mierzącej siłę powiązania par tekstów oznaczonych jako znaczeniowo podobne. Rzeczywista efektywność zależy jednak od charakterystyki podanych danych, przez co często wymagane jest dostosowanie modelu do szczególnego przypadku użycia. Modele pretrainede i dostosowane w technice Sentence-BERT na zbiorach danych dostosowanych do różnych zadań dostępne są do pobrania i swobodnego użytku na platformie Huggingface.

4.2.2 Wyszukiwanie podobieństw w przestrzeni wektorowej

Po przetworzeniu zbioru dokumentów za pomocą enkodera otrzymujemy zestaw wektorów składających się na wielowymiarową przestrzeń ukrytą. Na tym etapie pojawia się problem wydajnego wyszukiwania dokumentów kontekstowo spójnych z zadanym pytaniem. Treść zapytania zostaje

umieszczona w przestrzeni ukrytej poprzez zakodowanie dokładnie tą samą siecią, która posłużyła do przetworzenia pierwotnego zbioru danych. Dla wyszukiwania semantycznego stopień podobieństwa znaczeniowego między tekstami reprezentowanymi przez punkty może być określony za pomocą miar odległość między nimi. Do znalezienia dokumentów powiązanych na bazie odległości służą algorytmy wyszukiwania najbliższych sąsiadów.

Podstawową metodą rozwiązywania tego problemu, a zarazem zapewniającą największą trafność dla danego zbioru wektorów, jest wyszukiwanie wyczerpujące (ang. exhaustive). Polega ona na obliczeniu odległości między wektorem pytania a każdym innym elementem zbioru początkowego, co daje ranking podobieństw wszystkich dokumentów. Niestety dla zbiorów dokumentów o nawet niedużej liczności jest to procedura czasochłonna i kosztowna obliczeniowo. Aby osiągnąć wydajność akceptowalną dla systemu odpowiadającego użytkownikowi na bieżąco, koniecznym jest skorzystanie z przybliżonych metod wyszukiwania.

4.2.3 Indeksowanie przestrzeni wektorowej

Metodami cechującymi się znacznie mniejszą złożonością obliczeniową, w porównaniu do wyszukiwania wyczerpującego, są algorytmy k najbliższych sąsiadów opierające się o wcześniejsze indeksowanie przestrzeni ukrytej. Kosztem znacznego skrócenia czasu odnalezienia pobliskich elementów jest jednak otrzymanie lokalnego przybliżenia wyniku, wykorzystując wybrany sposób budowy indeksu.

FAISS

FAISS (Facebook AI Research Simliarity Search) [JDJ17] jest biblioteką zawierającą algorytmy wydajnego wyszukiwania podobieństw. Została napisana przez pracowników firmy Meta (wcześniej Facebook) na bazie wielu prac naukowych w dziedzinie wyszukiwania informacji opublikowanych na przestrzeni zeszłych dwóch dekad. Autorzy przygotowali implementacje selekcji metod przybliżonego wyszukiwania k najbliższych sąsiadów cechujących się kompromisem między trafnością zwróconych wyników a zaoszczędzeniem zasobów pamięciowych i skróceniem czasu wyszukiwania.

Dla danego zbioru wektorów o wymiarze d tworzy się strukturę indeksu wybranego typu. Typy indeksów są przystosowane do różnych zastosowań i różnią się cechami takimi jak optymalna ilość wymiarów i elementów w zbiorze, objętość pamięciowa struktury, a także precyzja oraz szybkość zwrócenia wyniku wyszukiwania. Do indeksu następnie dodaje się bazę wektorów, co rozpoczyna proces jego budowy. Funkcja wyszukiwania najbliższego sąsiada dla wektora x wykonuje poniższą operację:

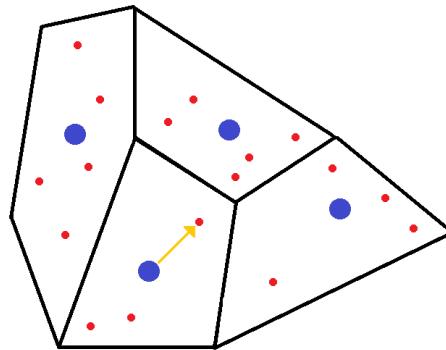
$$i = \operatorname{argmin}_{i=0:n} L(x, y_i)$$

gdzie L to wybrana miara odległości, a więc dla naszego przypadku podobieństwo cosinusowe opisane w 4.2.4, n to ilość przeszukiwanych wektorów, a y_i to kolejny wektor do zmierzenia odległości. W algorytmie wyczerpującym L to odległość Euklidesowa, a n to ilość wszystkich wektorów w bazie.

Przechodząc do zastosowanych w indeksach optymalizacji, podstawową techniką przybliżonego przeszukiwania przestrzeni jest struktura *indeksu odwroconego* (ang. inverted file index), w której wektory w bazie mapowane są na wcześniej wydzielone obszary przestrzeni ukrytej. Podział, będący pierwszym etapem kwantyzacji, następuje na etapie dostarczenia bazy do zaindeksowania za pomocą algorytmu grupowania k-srednich (ang. k-means), a każdy obszar nazywany *komórką Voronoi* (ang. Voronoi cell) reprezentowany jest przez centroid. Algorytm najpierw wyszukuje

centroid charakteryzujący obszar, w którym znajduje się zwięktryzowane pytanie, a odległości od poszczególnych elementów w bazie wyliczane są już tylko w tej komórce.

Kolejną optymalizacją jest dalszy podział przy pomocy *kwantyzacji wektorowej* (ang. product quantization, w skrócie PQ)[JDS11], która polega na zakodowaniu wektorów w skompresowanej formie dla danej *komórki Voronoia*. Tu algorytm k-średnich również znajduje zastosowanie, lecz tym razem obszary i centroidy określa się na podprzestrzeniach powstałyzych z mniejszych wektorów, uzyskanych przez podział oryginalnych wektorów z bazy na ustaloną ilość segmentów. Precyzujeć, na każdy segment przypada podprzestrzeń złożona z podwektorów o długości równej wymiarowi oryginalnych wektorów podzielonej przez ilość segmentów. Dzięki temu, do scharakteryzowania każdego elementu z bazy indeks przehowa jedynie kod składający się z identyfikatorów centroidów w każdej podprzestrzeni segmentu, co znacznie zaoszczędza pamięć zajmowaną przez strukturę. Objętość pamięciowa jest zależna od wybranej ilości segmentów — im więcej ich jest, tym dłuższe otrzymujemy słowo kodujące element bazy, ale również większą granularność zbioru i precyzję wyników wyszukiwania. Na koniec, aby móc zastosować wyszukiwanie po wektorach znajdujących się w obszarze *kwantyzacji wektorowej*, wyliczamy wektory szczegółowe (ang. residual vector) oryginalnych podwektorów do centroidów w podprzestrzeniach. Wektor szczegółowy stanowi różnicę centroidu od wektora znajdującej się w przestrzeni.



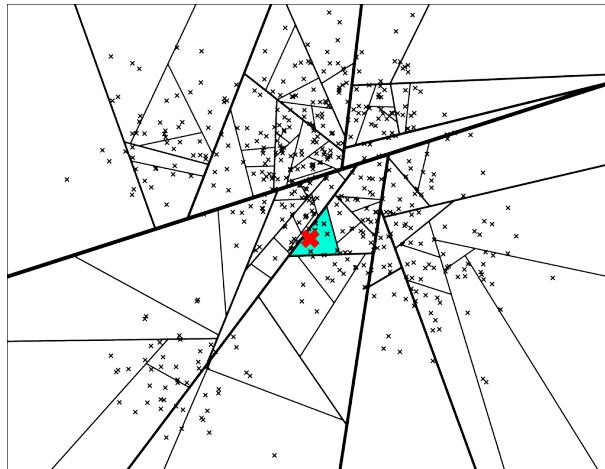
RYSUNEK 4.2: Fragment obszaru podprzestrzeni wektorowej podzielonej na *komórki Voronoia*. Żółtym kolorem zaznaczono wektor szczegółowy wyliczony pomiędzy centroidem obszaru (niebieski punkt) oraz punktem danych reprezentującym wyszukiwany dokument tekstowy (czerwony punkt).

Podczas wyszukiwania, element z zapytania zostanie zaklasyfikowany do komórki z pierwszego etapu podziału, czyli indeksu odwróconego. Następnie wektor podzielony zostanie na takie same segmenty jak w *kwantyzacji wektorowej*, a dla każdej z par fragmentu wektora zapytania i wektora szczegółowego w podprzestrzeni wyliczona zostanie kwadratowa odległość Euklidesowa. Do wynikowej struktury zostaną dodane tylko te odległości, które są mniejsze niż aktualnie minimalna wartość. W celu zwiększenia precyzji wyszukiwania procedurę powtórzyć można na kilku sąsiednich komórkach. Zapobiegnie to sytuacji, w której wyliczone zostaną odległości jedynie w obrębie jednego z obszarów, gdy wektor zapytania umieszczony jest w przestrzeni ukrytej na granicy komórek.

ANNOY

ANNOY (Approximate Nearest Neighbors Oh Yeah) to algorytm wymyślony przez Erika Bernhardssona wykorzystywany przez platformę Spotify do rekomendacji muzyki. Został on stworzony do przeszukiwania wielowymiarowych danych. Autorzy zaznaczają, że działa najlepiej, gdy ich wymiarowość nie jest zbyt duża (około 100 wymiarów), jednak algorytm potrafi działać zaskakująco dobrze nawet dla danych do 1000 wymiarów [Ber22]. ANNOY wyszukuje dane prawie tak samo szybko, jak najszybsze algorytmy aproksymacyjne wyszukiwania k najbliższych sąsiadów. Na tle innych algorytmów wyróżnia go korzystanie ze statycznych plików jako indeksów, co umożliwia dzielenie stworzonego indeksu między procesami oraz powoduje małe zużycie pamięci. Dodatkowo wykorzystanie plików tylko do odczytu oddziela tworzenie indeksu od wczytywania go, dzięki czemu możliwe jest rozpowszechnienie indeksu między urządzeniami i szybkiego wczytania, ale skutkuje to tym, że po zbudowaniu nie można dodać do niego więcej elementów.

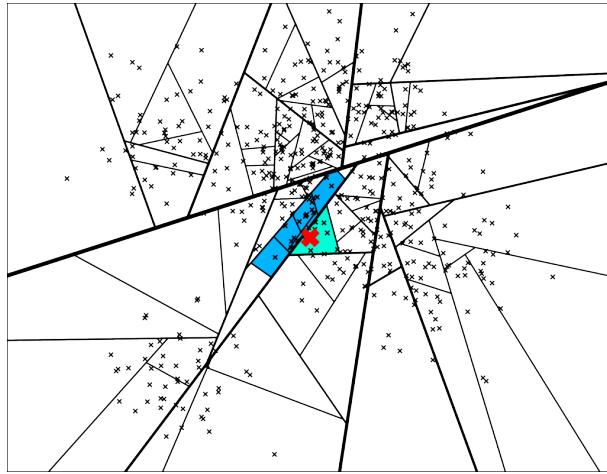
Działanie algorytmu ANNOY jest oparte na tworzeniu losowych projekcji przy tworzeniu drzew binarnych. Wybiera on w sposób losowy 2 punkty, a potem nakreśla hiperpłaszczyznę równoodległą od tych punktów, która dzieli przestrzeń na dwie podprzestrzenie. Krok ten jest rekurencyjnie powtarzany dla każdego węzła, aż w każdym węźle (podprzestrzeni) znajduje się co najwyżej N punktów. Mając tak zbudowane drzewo binarne, można wyszukać dany punkt idąc od korzenia drzewa w głąb lewego bądź prawego dziecka danego węzła. Wybór węzła jest prosty, gdyż wystarczy wybrać podprzestrzeń, w której szukany punkt się znajduje, aż do odnalezienia liścia (węzła, który nie dzieli się na mniejsze węzły). W ten sposób osiągnięto logarytmiczną złożoność czasową wyszukiwania, jednak należy pamiętać, że w danym liściu znajduje się maksymalnie N kandydatów do sprawdzenia.



RYSUNEK 4.3: Reprezentacja indeksu w dwuwymiarowej przestrzeni wyszukiwania kandydatów wraz z oznaczoną wybraną podprzestrzenią do przeszukania. [Ber15]

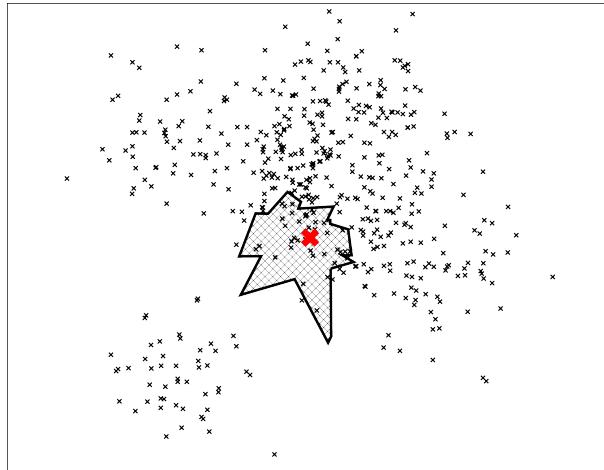
Wyszukiwanie za pomocą tylko jednego liścia nie jest idealnym rozwiązaniem, ponieważ poszukując k najbliższych sąsiadów, może się okazać, że k jest większe od liczby punktów zawartej w podprzestrzeni. Na dodatek istnieje możliwość, że najbliżsi sąsiedzi będą w sąsiedniej podprzestrzeni. ANNOY w swojej implementacji stosuje 2 metody zwalczania tego ograniczenia.

Pierwszym rozwiązaniem tego problemu jest przeszukiwanie węzłów, w których szukany punkt nie występuje, ale odległość do niego mieści się w pewnym progu akceptacji. Powoduje to odwiedzenie innych podprzestrzeni w poszukiwaniu najbliższych punktów. Należy wspomnieć, że ilość przeszukiwanych podprzestrzeni jest związana z ustaloną progiem akceptacji, (patrz rysunek 4.4).



RYSUNEK 4.4: Reprezentacja indeksu w dwuwymiarowej przestrzeni wyszukiwania kandydatów wraz z oznaczoną bazową podprzestrzenią do przeszukania (kolor zielony), poszerzoną o sąsiednie regiony. (kolor niebieski) [Ber15]

Drugim sposobem niwelowania tego problemu jest utworzenie lasu wielu drzew. Biorąc pod uwagę, że algorytm opiera się na tworzeniu losowych projekcji, każde drzewo będzie się od siebie różnić. Nakładając na siebie wszystkie drzewa, tworzymy sumę zbiorów punktów podprzestrzeni (patrz rysunek 4.5).



RYSUNEK 4.5: Reprezentacja indeksu w dwuwymiarowej przestrzeni wyszukiwania kandydatów wraz z oznaczoną sumą wybranych regionów wszystkich drzew [Ber15]

W wyznaczonej w ten sposób podprzestrzeni za pomocą jednej z opisanych w kolejnym podrozdziale miar obliczamy odległość każdego punktu od szukanej wartości i tworzymy ranking kandydatów.

Wyznaczanie podprzestrzeni kandydatów do przeszukania skutkuje znaczącym wzrostem wydajności z racji na mniejszą liczbę kosztownych czasowo operacji związanych z obliczaniem odległości. Dla porównania algorytm k najbliższych sąsiadów oblicza je dla każdego punktu osobno. Podejście to jest aproksymacyjne, czyli możliwe jest dobranie gorszej jakości kandydatów, niż stosując wyszukiwanie wyczerpujące.

ANNOY posiada parametry, dzięki którym można polepszyć jakość wyszukiwania kosztem wydajności. Pierwszym z nich jest ilość drzew, które chcemy stworzyć. Im więcej drzew tym większe prawdopodobieństwo, że dostaniemy podprzestrzeń z korzystnym podziałem. Jedynym ogranicze-

niem przy ciągłym zwiększaniu ilości drzew jest ilość pamięci, jako że wraz ze wzrostem liczby drzew rośnie nam rozmiar indeksu. Drugim parametrem, który algorytm pozwala nam zmienić, jest liczba przeszukanych kandydatów podczas przeszukiwania drzewa. Wraz ze wzrostem tego parametru algorytm będzie w każdym drzewie powiększał podprzestrzeń o sąsiednie podprzestrzenie, aż do pojawienia się żądanej liczby kandydatów. Zwiększenie tego parametru spowoduje wzrost jakości znalezionych kandydatów, kosztem wydajności algorytmu.

4.2.4 Miary odległości

Istnieje wiele miar odległości, które można wybrać do określenia podobieństwa między dokumentami [Cha07]. Wybrane z nich opisano poniżej.

Odległość Euklidesowa

Odległość Euklidesowa, również nazywana Pitagorejską, jest jedną z mniej skomplikowanych i prostych do zinterpretowania miar odległości. Mając dwa wektory reprezentujące punkty p_i oraz q_i w przestrzeni wielowymiarowej, wyliczana jest długość odcinka poprowadzonego pomiędzy nimi na podstawie współrzędnych Kartezjańskich:

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \quad (4.3)$$

Występuje także w wariantie zwany Kwadratem Odległości Euklidesowej, który przypisuje większą wagę obiektom bardziej oddalonym.

$$d^2(p, q) = \sum_{i=1}^n (p_i - q_i)^2 \quad (4.4)$$

Odległość Euklidesowa sprawdza się dobrze jedynie w zadaniach dotyczących przestrzeni o niskiej wymiarowości, ponieważ z każdym dodatkowym wymiarem miara zwiększa znacznie swój zakres, co utrudnia zróżnicowanie wyników. Ponieważ posługujemy się *enkoderami* tworzącymi reprezentację tekstu w postaci kilkuset liczb, do naszego przypadku koniecznym jest poszukanie alternatywnej miary.

Podobieństwo kosinusowe

Podobieństwo kosinusowe definiuje się jako kosinus kąta między wektorami p i q w przestrzeni iloczynów skalarnych. W zasadzie miara ta pozwala nam na określenie podobieństwa skierowania wektorów, co dostarcza nam cenną informację w przestrzeniach, w których prosta odległość może dawać niezadowalające wyniki.

$$\cos(p, q) = \frac{p \cdot q}{\|p\| \|q\|} = \frac{\sum_{i=1}^n p_i q_i}{\sqrt{\sum_{i=1}^n p_i^2} \sqrt{\sum_{i=1}^n q_i^2}} \quad (4.5)$$

Rozdział 5

Modele języka

Modele generatywne oparte na dekoderze w architekturze Transformerze są uczone w sposób samonadzorowany (ang. self-supervised) na dużych zbiorach danych tekstowych. Stosuje się do tego zadanie przewidywania następnego tokenu, będącego kontynuacją sekwencji podanej na wejściu kierując się minimalizacją funkcji entropii krzyżowej:

$$\mathcal{L} = - \sum_t \log p_\theta(x_t | x_{<t}) \quad (5.1)$$

Wynikiem predykcji ostatniej warstwy modelu jest funkcja logit o dla każdego tokenu w słowniku V , gdzie prawdopodobieństwo wyboru tokenu na i-tej pozycji z uwzględnieniem parametru temperatury T wyraża się, stosując funkcję softmax:

$$p_i \propto \frac{\exp(o_i/T)}{\sum_j \exp(o_j/T)} \quad (5.2)$$

Ostateczny wybór tokenu odbywa się przez próbkowanie z rozkładu prawdopodobieństwa tokenów ze słownika metodą:

- zachłanną (argmax) [RWC⁺18]
- temperature sampling [AHS85, FG17]
- algorytmu wyszukiwania najlepszego dopasowania (ang. beam search)
- *top-k* sampling [FLD18, HBF⁺18]
- nucleus sampling [HBD⁺20]

5.1 GPT

Generative Pre-trained Transfomers są rodziną autoregresywnych pretrenowanych modeli generatywnych stworzonych przez OpenAI. Aktualnie istnieje kilka wariantów, których budowa oparta jest o architekturę transformera, a dokładniej o bloki dekoderów. Wraz z rozwojem kolejnych wersji modyfikowano ich strukturę oraz wykorzystywano coraz większe zbiory treningowe, co znacznie poprawiło jakość generowanego tekstu.

5.1.1 GPT-3

GPT-3 [BMR⁺20] to największy z aktualnie udostępnionych przez OpenAI modeli i posiada aż 175 miliardów parametrów (wag połączeń) oraz 96 warstw. Jego architektura nieznacznie różni się

od poprzednika tj. GPT-2 [RWC⁺18], z wyjątkiem alternatywnego doboru gęstych i rzadkich wzorców samouwagi pozwalających, m.in. zmniejszyć złożoność obliczeniową, o których wspominają autorzy. Warto także zwrócić uwagę, iż umożliwiło to, by oryginalnie okno kontekstowe przyjmowało do 2048 tokenów.

Zdolność modelu do wykonywania wielu zadań kilkukrotnie lepiej od swoich konkurentów osiągnięto, m.in. trenując GPT-3 na specjalnie przygotowanym ogromnym zbiorze danych składającym się w głównej mierze z uprzednio przefiltrowanego zbioru CommonCrawl wraz z dodanymi stosunkowo mniejszymi zbiorami WebText, Wikipedia oraz korpusu książek Books1 i Books2. Twórcy w swojej pracy zawarli również informację, że ich model pracuje najlepiej w konwencji few-shot, która polega na dodaniu kilku przykładów do zadania, aby mógł on przystosować swoje działanie i wygenerować najbardziej oczekiwana sekwencję tokenów. Pomimo tego potrafi także rozwiązywać zadania bez podania wcześniejszych przykładów, czyli w konwencji 0-shot.

5.1.2 GPT-J

W odpowiedzi na komercyjne modele języka naturalnego tworzone przez OpenAI powstał konkurencyjny projekt open-source prowadzony przez EleutherAI. Autorzy, wzorując się na architekturze GPT-3 w wersji z 6.7 miliardami parametrów (wag połączeń), stworzyli własny model GPT-J z 6 miliardami parametrów [Wan21]. Twórcy zdecydowali się na dodanie kilku ulepszeń architektonicznych, m.in. zastosowanie enkodera pozycyjnego RoPE [SLP⁺21] i równolegle umieszczenie warstwy samouwagi z warstwą sieci neuronowej.

GPT-J został wytrenowany na zbiorze danych The Pile, który zawiera 800Gb wysokiej jakości korpusów tekstowych. Oznacza to, że połączone zbiory danych zostały przygotowane w taki sposób, aby nie zawierały duplikatów, charakteryzowały się poprawnością i wysoką wartością informacji oraz nie posiadały pozostałości wynikających z ich źródła pochodzenia np. znaczników HTML stron internetowych. Jego struktura oparta jest o dokumenty z wielu kategorii takich jak internet lub społeczność akademicka. Ich analiza wskazuje na pochodzenie zarówno z ArXiv, FreeLaw, specjalnie przygotowanego zbioru z CommonCrawl, Wikipedii oraz Github. Ogromna różnorodność z jednocośną wysoką jakością danych pozwoliła, aby wytrenowany model osiągał podobne wyniki jak wspomniany wcześniej GPT-3 w wersji z porównywalną liczbą parametrów.

5.2 BLOOM 176B

Empiryczne obserwacje dotyczące prawa skalowalności dużych modeli [KMH⁺20] doprowadziły do wyraźnego zainteresowania badaczy stosowaniem coraz większych architektur specjalizowanych dla przetwarzania języka naturalnego. Prawo to mówi, iż wraz ze wzrostem liczby parametrów, wielkość zbioru danych uczących oraz wykorzystywanych zasobów mocy obliczeniowej wzrasta również jakość działania modelu. Niestety wymagania sprzętowe oraz finansowe trenowania tak wielkich modeli, jak np. GPT-3 są osiągalne jedynie dla dużych korporacji, które niechętnie je udostępniają.

Powyższe fakty stały się motywacją dla organizacji BigScience, która inicjując wspólne badania wielu ekspertów, stworzyła wielojęzyczny model ze 176 miliardami parametrów (wag połączeń), a następnie udostępniła go do wolnego i powszechnego użytku.

BLOOM [SFA⁺22], gdyż tak nazwano powstały projekt, oparty jest o architekturę dekodera znaną z transformerów. Ostateczna konfiguracja składa się z warstwy zanurzenia (ang. input embedding), 70 warstw bloków transformera oraz warstwy modelującej wygenerowane słowa. Ważnym elementem architektury jest tokenizator, który buduje słownik składający się z 250 tys. wyrazów,

a jego działanie oparto na Byte Pair Encoding. Dodatkowo na początku przeprowadzana jest pretokenizacja, która ma na celu, m.in. ograniczyć maksymalną długość wytwarzanej sekwencji. Jeśli mowa o blokach transformerów, wykorzystano wspomnianą wcześniej część dekodera ze 112 głowicami samouwagi i zaproponowano nowe rozwiązania. Warto zwrócić uwagę na zastosowanie kodowania pozycyjnego ALiBi (ALiBi positional embedding), które zamiast dodawać informację o pozycji już na etapie warstwy osadzenia (ang. embedding layer), osłabiają wynik uwagi w zależności od odległości kluczy i zapytań w warstwie samouwagi. Innym usprawnieniem okazała się także dodatkowa warstwa normalizująca umieszczona bezpośrednio po pierwszej warstwie osadzenia, która gwarantuje stabilność uczenia wag połączeń modelu kosztem karania generalizacji podejścia 0-shot.

W celu stworzenia modelu języka ogólnego zastosowania autorzy wykorzystali zbiór treningowy ROOTS, który zawiera w sobie aż 46 różnych języków naturalnych oraz 13 języków programowania zawartych w zebranych dokumentach oraz repozytoriach kodu. Korpus posiada 350 miliardów tokenów pochodzących z trzech różnych źródeł, które dzięki swojej specyfice zapewniają odpowiednią stymulację w trakcie treningu. Największa część zbioru przygotowano zachęcając uczestników specjalnych warsztatów do kolekcjonowania źródeł pokrywających dużą część wybranych języków. Dodatkowo zostali poproszeni o wskazanie znaczących w ich lokalizacji stron internetowych, które dodano używając pseudocrawl. Następnie korpus został wzbogacony o zbiór składający się z języków programowania ze strony GitHub. Ostatecznie całość dopełnił OSCAR, który jest zbiorem przetworzonych dokumentów zebranych do lutego 2021 roku przez organizacje Common Crawl¹.

Model został wytrenowany na superkomputerze Jean Zey przy wykorzystaniu 384 procesorów graficznych A100 z 80GB pamięci każda. Stabilność i szybkość zapewnił framework Megatron-DeepSpeed, który pozwolił na połączenie trzech komplementarnych podejść do uczenia rozproszonego:

- **równoległość danych** - zreplikowano model 8 razy, a każda z kopii otrzymała osobny fragment korpusu treningowego, by synchronizować wykonaną pracę z pozostałymi kopiami po każdym pojedynczym kroku treningowym.
- **równoległość tensorów** - poszczególne warstwy modeli zostały podzielone na kilka urządzeń, dzięki czemu, zamiast mieć cały tensor na pojedynczym procesorze graficznym, jego części składowe są rozlokowane na kilku.
- **równoległość potokowa** - warstwy modeli rozlokowano na wiele procesorów graficznych, dzięki czemu tylko część z warstw znajduje się na każdym procesorze.

Pomimo wielkości modelu, zastosowane rozwiązania pozwalają na wygenerowanie wyniku uruchomiając go na komputerach z około 700GB wolnej przestrzeni dyskowej oraz jedynie 16GB pamięci RAM. Jest to możliwe, m.in. dzięki temu, że architektura transformera pozwala na załadunek w danym momencie pojedynczego, aktualnie wymaganego bloku. Udostępniono również publiczne API poprzez platformę huggingface.co.

5.3 Cohere xlarge

Rozpowszechnienie zastosowań modeli języka naturalnego w wielu komercyjnych zadaniach takich jak ekstrakcja informacji, czy moderowanie zawartości artykułów i stron internetowych,

¹<https://commoncrawl.org/> - Common Crawl jest organizacją, która bezpłatnie udostępnia repozytoria dokumentów zebranych w trakcie regularnego przeszukiwania informacji w Internecie

pozwoliło na rozwój nowych organizacji. Przykładem jest Cohere oraz ich własna rodzina modeli języka naturalnego, które służą do generowania tekstu.

Model xlarge jest aktualnie największym z udostępnianych przez Cohere wariantów. Posiada on 52.4 miliardów parametrów (wag połączeń) i został stworzony w architekturze generatywnego pretrenowanego transformera. Do wyuczenia wykorzystano autorski zbiór danych anotowany jako coheretext-filtered, lecz o jego zawartości i strukturze wiadomo niewiele. Podano jednak informacje, iż sam model udostępnia generowanie tekstu jedynie w języku angielskim, a także, iż okno kontekstowe przyjmuje do 2048 tokenów. Twórcy nie udostępniają niestety szczegółowych informacji dotyczących budowy modelu, lecz zwracają uwagę na potrzebę świadomości zagrożeń związanych z wykorzystaniem generatywnych modeli języka.

Wariant xlarge udostępniono w formie API, które można konfigurować oraz odpytywać po zdobyciu klucza. Jego ostatnia aktualizowana wersja, wykorzystywana również w pracy, pochodzi z listopada 2022 roku.

W zaproponowanym systemie zdecydowano się na wykorzystanie pretrenowanych modeli generatywnych języka naturalnego BLOOM oraz Cohere xlarge, ze względu na udostępnione przez twórców darmowe API, które pozwala na odpytywanie ich bez ograniczeń wynikających z posiadanej lokalnie mocy obliczeniowej i zasobów pamięciowych. Dodatkowym czynnikiem jest także fakt, iż dotychczas eksperymenty z udziałem tych modeli rzadko pojawiają się w literaturze naukowej.

Rozdział 6

In-context learning — uczenie w kontekście

Uczenie w kontekście (ang. in-context learning) jest zjawiskiem wyłaniającym się w generatywnych modelach języka takich jak GPT-3, w którym ten model dostosowuje swoją predykcję do sformułowanego zadania oraz podanych przykładów (ang. few-shot learning) w języku naturalnym w trakcie przejścia sygnału przez sieć (ang. forward pass). W przeciwieństwie do uczenia nadzorowanego lub nienadzorowanego nie dochodzi do aktualizacji wag połączeń modelu. Zastosowanie tej metody sprawia, że nie potrzeba go dostosować do nowych zadań, co redukuje koszt obliczeniowy i wpływa na zmniejszenie potrzebnej ilości danych do kilku przykładów uczących.

Nietypowość uczenia w kontekście wynika z wyłaniania się tego zjawiska w dużych modelach języka, gdyż nie jest to zamiar procesu ich pretrenowania. Dodatkowo forma reprezentacji danych korpusów tekstowych, które zazwyczaj są stronami internetowymi lub fragmentami książek i artykułów, znacznie różni się od formy promptów ujawniających te zdolności, które są połączeniem instrukcji z przykładami zadań. Przyczyna wyłaniania się tego fenomenu nie jest jeszcze znana. Dotychczasowe próby formalizacji procesu uczenia w kontekście opierają się na hipotezach wskazujących na jego podobieństwo do teorii *wnioskowania bayesowskiego* [XRLM21] lub *meta-uczenia* (ang. meta-learning) [vONR⁺22, DSD⁺22, ASA⁺22]. Przeprowadzone eksperymenty pokazują, że zdolność modelu do uczenia się w kontekście zależy od liczby parametrów [KMH⁺20] oraz rozkładu danych uczących zgodny z prawem Zipfa [CSL⁺22].

6.1 Prompting

Prompt jest wzorcem zawierającym instrukcję i (lub) przykłady wskazujące na specyfikę zadania, podawanym jako wejście, aby model, przez uczenie w kontekście, dostosował do niego swoją odpowiedź. Instrukcja jest jego opisem, np.: "Odpowiedz na pytanie" albo "Zaklasifykuj wydruk podanych zdań jako pozytywny, neutralny lub negatywny", natomiast przykłady to pary wejścia-wyjścia, czyli wzory zadania z jego poprawnymi rozwiązaniami. Standardowym sposobem promptingu jest *few-shot prompting* wprowadzony w [BMR⁺20]. Liczba przykładów jest ustalana, biorąc pod uwagę ograniczenie długości okna kontekstowego modelu oraz liczbę tokenów w przykładach. Odpowiednik wzorca bez przykładów, tzw. 0-shot zawiera tylko instrukcję opisującą zadanie. Często przy ocenie modelu na zbiorze danych prompt 0-shot stanowi punkt odniesienia dla bardziej rozbudowanych wzorców.

TABELA 6.1: prompt 3-shot w zadaniu tłumaczenia — przykład z [BMR⁺20]

Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush giraffe => girafe peluche
cheese =>

W tabeli 6.1 pokazano prompt dla zadania tłumaczenia z języka angielskiego na język francuski z podanymi trzema przykładami oraz docelowym przykładem do przetłumaczenia przez model języka. W analogiczny sposób wiele zadań przetwarzania języka naturalnego może zostać przeformuowane jako zadania kontynuacji sekwencji, co czyni modele języka wielozadaniowymi.

Warto wspomnieć, że istnieje wariant promptingu wykorzystujący *soft-prompts* [LARC21] pełniący funkcję prefiksowych tokenów dodawanych do tekstu wejściowego. W praktyce te tokeny wyuczają się algorytmem propagacji wstecznej bez aktualizacji wag połączeń modelu języka w procesie dostrajania modelu do danego zadania, zwanym *p-tuning*. W ramach tej pracy skupiliśmy się jednak na promptach zapisanych w języku naturalnym oraz *strategiach promptingu*, czyli metodach wykorzystujących je do wieloetapowego generowania kroków potrzebnych do rozwiązania zadania.

6.1.1 Prompting dynamiczny

Większość rozwiązań opiera się na niezmieniających się wzorach, jednak możliwe jest generowanie unikalnych promptów dla każdego przypadku. W promptingu dynamicznym konstruowany jest nowy prompt będący zbiorem k najbardziej podobnych przykładów ze zbioru treningowego do przypadku testowego. Do budowy wzorca można użyć tych samych metod stosowanych w wyszukiwaniu informacji, a zbiorem dokumentów stają się wtedy przykłady zbioru treningowego. Motywacją do stosowania takiego podejścia jest idea, która zakłada, że podobne przykłady ze zbioru treningowego mogą wpływać pozytywnie, dostarczając więcej informacji o przypadku testowym.

6.1.2 Chain of thought wnioskowania

Łańcuch wnioskowania (ang. chain of thought) to sekwencja pośrednich kroków rozumowania w języku naturalnym. Strategia ta ma za zadanie wymusić, aby wygenerowana odpowiedź modelu zawierała analogiczną argumentację, wynikającą ze wzmacniania podanych przykładów w promptcie o kolejne kroki wiodące do prawidłowej odpowiedzi [CND⁺22]. Dotychczasowe eksperymenty [WWS⁺22b] pokazują znaczący wpływ tej wersji strategii promptingu, która przewyższyła wyniki modeli języka dostrojonych do zadań arytmetycznych, np. na benchmarku GSM8K składającego się z zadań tekstowych z matematyki na poziomie szkoły podstawowej.

Takie podejście jest analogiczne do umożliwienia człowiekowi rozpisania problemu i kolejnych kroków rozwiązania na kartce papieru przed udzieleniem odpowiedzi.

TABELA 6.2: Łańcuch wnioskowania na GSM8K, 2-shot — przykład z [CND⁺22]

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?"
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9.

6.1.3 Self-ask

Self-ask jest strategią promptingu opisaną przez autorów pracy [PZM⁺22], która polega na dekompozycji złożonych pytań na prostsze pytania pośrednie. W ramach tej strategii proponowany jest prompt składający się z początkowego pytania wzbudzającego model do podjęcia decyzji, czy wymagany jest dalszy podział bazowego problemu na szczegółowe podpytania. Jeśli model wygeneruje pozytywną odpowiedź, wówczas wymagane jest także, aby samodzielnie stworzył pośrednie pytania, na które będzie musiał następnie odpowiedzieć. Tak zebrane częściowe informacje, mają za zadanie ułatwić wygenerowanie ostatecznej odpowiedzi na oryginalne pytanie. Przykład takiego promptu znajduje się w tabeli 6.3. Ograniczeniem tej strategii w wersji zaproponowanej przez jej autorów jest potrzeba dopasowania wzorców do specyficznego zbioru pytań, dla których zawsze wymagane są pytania pośrednie, co w przypadku pytań spoza zbioru się nie sprawdza.¹

TABELA 6.3: Self-ask, 1-shot — przykład z [PZM⁺22]

Question: Who lived longer, Muhammad Ali or Alan Turing?
Are follow up questions needed here: Yes.
Follow up: How old was Muhammad Ali when he died?
Intermediate answer: Muhammad Ali was 74 years old when he died.
Follow up: How old was Alan Turing when he died?
Intermediate answer: Alan Turing was 41 years old when he died.
So the final answer is: Muhammad Ali

6.1.4 ReAct

Do tej pory zdolności dużych modeli języka w problemach wnioskowania w języku naturalnym i planowania sekwencyjnych akcji [BBC⁺22] były badane oddzielnie. Google jako pierwszy przełamał tę barierę prezentując strategię promptingu nazwaną ReAct (ang. reasoning and acting) [YZY⁺22], która jest paradygmatem zapożyczającym terminologię z uczenia przez wzmacnianie (ang. reinforcement learning). Model języka staje się w nim agentem planującym sekwencję akcji za pomocą łańcuchów wnioskowania, natomiast środowiskiem jest system, z którym agent może wchodzić w interakcję. Generuje on komendy w języku naturalnym, wykonując pewne działanie, w rezultacie otrzymując informację zwrotną. Każda trajektoria takiej interakcji składa się z wielu etapów myśl-działanie-obserwacja (thought-action-observation).

Według autorów ReAct zmniejsza wpływ halucynacji wiedzy modeli języka w problemie udzielania odpowiedzi na pytanie przez interakcję z zewnętrznym źródłem faktów. Zbiorem, na którym

¹W tabeli A.6 na stronie 114 przedstawiamy użytą przez nas alternatywną wersję promptu czerpiącą inspirację ze strategii self-ask.

dokonano eksperymentów był HotpotQA, a środowiskiem odpowiedzialnym za dostarczanie agentowi informacji była Wikipedia, z którą można wejść w interakcję poprzez komendy *Search* i *Lookup*. Przykładowy prompt w tabeli 6.4 składa się z trajektorii będącej przeplotem myśli, akcji oraz obserwacji (zaznaczonych na niebiesko) w zadaniu udzielenia odpowiedzi na pytanie. Komenda *Search* zawierającą kwerendę wyszukuje ją w zbiorze dokumentów Wikipedii i wybiera najbardziej pasującą do niej stronę, z której posiłkuje się paroma zdaniami opisu jako obserwacja dla agenta. W przypadku nieznalezienia satysfakcyjnej odpowiedzi w poprzedniej obserwacji agent może użyć kwerendy *Lookup* do wyszukania zdań na wyselekcjonowanej stronie zawierających podane słowa w kwerendzie. Gdy według agenta możliwe jest rozwiązywanie problemu generuje on komendę *Finish* z kwerendą będącą ostateczną odpowiedzią i kończy swoje działanie.

TABELA 6.4: ReAct, 1-shot — przykład z [YZY⁺22]

Question: What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?

Thought 1: I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends into, then find the elevation range of the area.

Action 1: *Search[Colorado orogeny]*

Observation 1: The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding areas.

Thought 2: It does not mention the eastern sector. So I need to look up eastern sector.

Action 2: *Lookup[eastern sector]*

Observation 2: (Result 1 / 1) The eastern sector extends into the High Plains and is called the Central Plains orogeny.

Thought 3: The eastern sector of Colorado orogeny extends into the High Plains. So I need to search High Plains and find its elevation range.

Action 3: *Search[High Plains]*

Observation 3: High Plains refers to one of two distinct land regions

Thought 4: I need to instead search High Plains (United States).

Action 4: *Search[High Plains (United States)]*

Observation 4: The High Plains are a subregion of the Great Plains. From east to west, the High Plains rise in elevation from around 1,800 to 7,000 ft (550 to 2,130 m).[3]

Thought 5: High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft.

Action 5: *Finish[1,800 to 7,000 ft]*

Rozdział 7

Zbiory dokumentów i pytań

Dokument definiowany jest jako pewien materiał mający wartość informacyjną, a także w kontekście informatyki, jako spójna informacja przetwarzana przez system komputerowy [SJP22]. Zbiór dokumentów przechowywanych w ustrukturyzowany sposób tworzy bazę dokumentów, którą można wykorzystywać w różnych celach. Zebrane dane w postaci plików tekstowych, audio lub video bywają umieszczane w lokalnej przestrzeni dyskowej w formie nieprzetworzonych plików, a także jako bazy danych SQL lub noSQL. Obecnie bardzo częstym rozwiązań jest wykorzystywanie zbiorów znajdujących się w chmurach i udostępnionych publicznie za sprawą internetu wraz z dedykowanym API.

Poniżej przedstawiamy wykorzystywane przez nas narzędzie do modelowania tematycznego o nazwie BERTopic, następnie definiujemy benchmark wraz z powszechnymi miarami oceny, a zaraz po nim opisujemy i analizujemy wykorzystane przez nas zbiory pytań oraz dokumentów.

7.1 BERTopic

BERTopic [Gro22] jest techniką modelowania tematycznego, czyli odnajdywania abstrakcyjnych tematów występujących w zbiorze dokumentów. Wiele konkurencyjnych rozwiązań jak np. LDA (Latent Dirichlet Allocation)[BNJ03] opisuje dokumenty w postaci "worka słów", a następnie określa tematy na podstawie tak wybranych reprezentacji, co sprawia, że pomijane są znaczenia wyrazów wynikające z kolejności ich występowania. W przeciwieństwie do powyższego podejścia BERTopic enkoduje dokumenty wykorzystując do tego wspomnianą we wcześniejszym rozdziale architekturę sentence-transformer, która pozwala na stworzenie wektorowej reprezentacji dokumentów, biorąc pod uwagę kontekst, w jakim występują słowa.

Kolejnym krokiem po stworzeniu reprezentacji liczbowej jest redukcja ich wymiaru, aby umożliwić efektywne zastosowanie metody grupowania. Domyślnie stosuje się UMAP (Uniform Manifold Approximation and Projection) [MHM18], który potrafi zachować lokalną i globalną strukturę danych mimo zmniejszenia wielkości wektorów.

Następnie tak przygotowane dane są grupowane wybraną metodą grupowania. Autorzy podstawowo wskazują na użycie HDBSCAN [MH17], której zaletą jest odnajdywanie grup o różnych kształtach i możliwość identyfikowania wartości odstających.

Kolejnym etapem jest stworzenie "worka słów" dla każdej z powstałych grup. W tym celu łączone są wszystkie dokumenty w obrębie danego zbioru, a następnie zliczane występowanie poszczególnych wyrazów. Dodatkowo stosuję się normalizacje L1, aby zniwelować różnice wynikające z ich rozmiarów.

Ostatecznym krokiem jest określenie tematu dotyczącego danej grupy. Autorzy proponują zmo-

dyfikowaną metodę TF-IDF nazwaną *class-based TF-IDF*, która w przeciwieństwie do podstawowej wersji, zamiast porównywać ważność słów między pojedynczymi dokumentami, bierze pod uwagę wszystkie dokumenty w grupie połączone w całość. W ten sposób obliczana jest ocena ważności słów w grupie, a co za tym idzie, wybierane są najbardziej reprezentatywne wyrazy tworzące temat.

$$W_{x,c} = ||tf_{x,c}|| \cdot \log\left(1 + \frac{A}{f_x}\right) \quad (7.1)$$

x - słowo

c - klasa

$||tf_{x,c}||$ - częstotliwość występowania słowa x w klasie c z normalizacją L1

A - średnia liczba słów w klasie

f_x - częstotliwość występowania słowa x we wszystkich klasach

Opcjonalnie na końcu stosuje się metodę maksymalnej istotności marginalnej (ang. Maximal Marginal Relevance), która wyszukuje spójne słowa bez ich nadmiernego nakładania się między dokumentami, czyli usuwa takie, które nie wnoszą dodatkowych informacji do tematu.

7.2 Benchmark

W uczeniu maszynowym słowo benchmark wykorzystuje się w kontekście testowania, a także porównywania jakości oraz możliwości tworzonych rozwiązań. Można go traktować jako test, w ramach którego autorzy dostarczają zarówno zadania dotyczące konkretnego problemu, jak i miary oceny służące ocenie ich wykonania. Dziedzina przetwarzania języka naturalnego również posiada wiele benchmarków pozwalających sprawdzić jakość poszczególnych zastosowań m.in. wyszukiwania informacji czy odpowiadania na zadane pytania. Przykładem są opisane poniżej OpenBookQA [MCKS18] oraz HotpotQA [YQZ⁺18].

7.2.1 Miary oceny

Trafność

Trafność (ang. accuracy) jest stosunkiem poprawnych predykcji wyników do wszystkich testowanych

$$\text{Trafność} = \frac{\text{Liczba poprawnych wyników}}{\text{Liczba wszystkich wyników}} \quad (7.2)$$

Dokładne dopasowanie

Dokładne dopasowanie (ang. exact match) wynosi 1, gdy przewidziany łańcuch znaków jest identyczny co łańcuch referencyjny. W przeciwnym wypadku wartość jest równa 0.

$$\text{Dokładne dopasowanie} = \begin{cases} 1, & \text{predykcja} == \text{referencja} \\ 0, & \text{predykcja} != \text{referencja} \end{cases} \quad (7.3)$$

Pokrycie dokładnego dopasowania

Pokrycie dokładnego dopasowania (ang. cover exact match) wynosi 1, gdy przewidziany łańcuch znaków zawiera się w łańcuchu referencyjnym. W przeciwnym wypadku wartość jest równa 0.

$$\text{Pokrycie dokładnego dopasowania} = \begin{cases} 1, & \text{referencja.zawiera(predykcja)} \\ 0, & \text{! referencja.zawiera(predykcja)} \end{cases} \quad (7.4)$$

Miara F1

Miara F1 (ang. F1-measure) jest średnią harmoniczną liczoną według precyzji (ang. precision) i czułości (ang. recall). W przypadku zastosowania tej miary dla przetwarzania języka naturalnego, precyzja jest jako liczbą wspólnych tokenów wygenerowanej odpowiedzi z odpowiedzią referencyjną podzieloną przez liczbę wygenerowanych tokenów, natomiast czułość to liczba wspólnych tokenów wygenerowanej odpowiedzi z odpowiedzią referencyjną podzielona przez liczbę tokenów referencyjnych.

$$F1 = \frac{2 \cdot \text{precyzja} \cdot \text{czułość}}{\text{precyzja} + \text{czułość}} \quad (7.5)$$

$$\text{precyzja} = \frac{\text{Liczba wspólnych tokenów}}{\text{Liczba tokenów przewidzianych}} \quad (7.6)$$

$$\text{czułość} = \frac{\text{Liczba wspólnych tokenów}}{\text{Liczba tokenów referencyjnych}} \quad (7.7)$$

BLEU

BLEU (Bilingual Evaluation Understudy)[PRWZ02] jest algorytmem służącym do oceny jakości wygenerowanego tekstu. Oryginalnie służył do porównania przetłumaczonego fragmentu z zaproponowanymi tłumaczeniami referencyjnymi. Podstawa jego działania są operacje porównujące występowanie n-gramów w przewidzianym wyniku z podanymi referencjami. Liczy się go jako ważoną średnią geometryczną zmodyfikowanych precyzji n-gramów mnożoną przez karę za zwięzłość (ang. Brevity Penalty)

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right) \quad (7.8)$$

$$BP = \begin{cases} 1, & c > r \\ e^{\frac{1-r}{c}}, & c \leq r \end{cases} \quad (7.9)$$

$$p_n = \frac{\sum_{C \in \{\text{Candidates}\}} \sum_{n-gram \in \{C\}} Count_{clip}(n-gram)}{\sum_{C' \in \{\text{Candidates}\}} \sum_{n-gram' \in \{C'\}} Count(n-gram')} \quad (7.10)$$

$$Count_{clip} = \min(Count, \text{Max_Ref_Count}) \quad (7.11)$$

BP - kara za zwięzłość

r - długość korpusu referencyjnego

c - długość przewidzianego łańcucha

p_n - zmodyfikowana precyzja n-gramu

w_n - pozytywna waga

Count - liczba zaobserwowanego n-gramu

Max_Ref_Count - liczba zaobserwowanego n-gramu w łańcuchu referencyjnym

7.3 OpenBookQA

OpenBookQA [MCKS18] jest benchmarkiem stworzonym w ramach problemu odpowiadania na pytania wielokrotnego wyboru. Zawiera on zestaw 5,957 pytań w języku angielskim podzielonych na zbiór treningowy, walidacyjny oraz testowy wraz z podanymi czterema możliwymi odpowiedziami i wybraną poprawną z nich. Pytania mają charakter naukowy dotyczący elementarnej wiedzy z

różnych dziedzin. Przykładowe pytania zawarte benchmarku zostały umieszczone w tabeli 7.1. Autorzy udostępnili dodatkowo zbiór 5,167 faktów, które zawierają informacje konieczne do udzielenia poprawnej odpowiedzi, lecz jak wspominają, wymagana jest też wiedza dodatkowa spoza ich zakresu. Wspomniane fakty można zdefiniować jako jedno krótkie zdanie zawierające pewną informację np. "A bee is a pollinating animal". Do ewaluacji oryginalnie stosowana jest trafność, dla której punktem odniesienia sztucznych sieci neuronowych jest 50%, przy czym średnim wynikiem człowieka jest 92%.

TABELA 7.1: Przykładowe pytania z benchmarku OpenBookQA

Pytanie	Odp A	Odp B	Odp C	Odp D	Poprawna Odpowiedź
There is most likely going to be fog around:	a marsh	a tundra	the plains	a desert	A
A cactus stem is used to store	fruit	liquid	food	spines	D
Which requires energy to move?	weasel	willow	mango	poison ivy	A
An animal that only eats plants is a	rat	moth	chimpanzee	pig	B

7.3.1 Analiza dokumentów

Korzystając ze wspomnianego narzędzia modelowania tematycznego BERTopic przeprowadziliśmy analizę tematyczną dokumentów zawartych w benchmarku OpenBookQA, a uzyskane wyniki przedstawiliśmy w tabeli 7.2. Można zauważyć, że otrzymane grupy tematyczne zawierają słowa kluczowe związane z szeroko pojętymi naukami przyrodniczymi.

Najliczniejszą grupą okazał się zbiór oznaczony indeksem -1 nazwany *-1_environment_soil_water_increases*, który przedstawia elementy odstające (ang. outliers). W tabeli 7.3 widoczne jest 10 najlepszych słów opisujących tę grupę oraz ich poszczególna ilość w dokumentach. Wynika z niej, że pomimo uznania elementów z tej grupy za odstające, jej słowa kluczowe nadal związane są z naukami przyrodniczymi.

7.3.2 Analiza pytań

Analizę pytań testowych przeprowadziliśmy w ten sam sposób co analizę dokumentów. Uzyskane wyniki przedstawiliśmy w tabeli 7.4, z której wywnioskować można, że podobnie jak dokumenty, pytania związane są z naukami przyrodniczymi. Dodatkowo elementy odstające przedstawiliśmy w tabeli 7.5. Można zauważyć, że 159 pytań należących do tej grupy, charakteryzuje się dużą różnorodnością, gdyż jak wskazują wartości we wspomnianej tabeli, 10 najczęściej występujących słów pojawia się w małej części pytań odstających. Warto zauważyć, iż pytania w benchmarku mają różną konstrukcję i mogą polegać m.in. na wybraniu zakończenia zdania, wybraniu poprawnej odpowiedzi na postawione pytanie, lub wybraniu odpowiedniej tezy spośród 4 opcji.

TABELA 7.2: Wyniki analizy tematycznej dokumentów z banchmarku OpenbookQA uzyskane za pomocą BERTopic

Temat	Liczba dokumentów	Nazwa Tematu
-1	293	-1_environment_soil_water_increases
0	97	0_formed_rocks_rock_erosion
1	67	1_resource_pollution_renewable_nonrenewable
2	65	2_inherited_characteristic_offspring_eggs
3	65	3_eat_prey_predators_insects
4	62	4_electrical_electricity_circuit_conductor
5	36	5_vapor_clouds_water_precipitation
6	35	6_seeds_seed_flower_dispersal
7	34	7_heat_hot_cool_substance
8	34	8_habitat_humans_ecosystem_environment
9	32	9_photosynthesis_dioxide_carbon_cell
10	31	10_axis_day_sun_earth
11	31	11_measure_measuring_used_speed
12	28	12_magnetic_compass_magnetism_metals
13	25	13_moon_surface_phases_craters
14	25	14_plant_stem_roots_soil
15	25	15_light_object_reflects_reflect
16	24	16_water_lives_aquatic_ocean
17	24	17_sound_string_vibrate_vibrating
18	23	18_friction_object_force_speed
19	22	19_sunlight_solar_sun_nuclear
20	21	20_nutrients_requires_animal_food
21	19	21_hemisphere_winter_summer_daylight
22	18	22_point_solid_liquid_freezing
23	18	23_population_species_decreases_prey
24	18	24_warm_temperatures_polar_shiver
25	17	25_telescope_microscope_observing_eyes
26	16	26_pressure_altitude_available_increases
27	16	27_tree_trees_cutting_planting
28	15	28_oxygen_lungs_circulatory_blood
29	15	29_chemical_substances_substance_change
30	14	30_camoouflage_hiding_color_predators
31	13	31_matter_phase_variable_shape
32	13	32_bacteria_skin_pasteurization_poisoning
33	13	33_producer_role_chain_food
34	12	34_impact_health_positive_negative
35	10	35_candle_burning_forest_living

TABELA 7.3: 10 najczęstszych słów kluczowych i liczbą dokumentów, w których występują w grupie dokumentów odstających

Słowo klucz	Liczba dokumentów
environment	72
soil	31
water	104
increases	62
source	101
food	63
gas	23
organism	65
decreases	27
desert	9

TABELA 7.4: Wyniki analizy tematycznej pytań z banchmarku OpenbookQA uzyskane za pomocą BERTopic

Temat	Liczba pytań	Nazwa Tematu
-1	159	-1_desert_rabbit_dog_live
0	60	0_evaporation_water_canyon_fuels
1	53	1_animal_predator_eat_animals
2	47	2_car_electricity_electric_runs
3	44	3_plants_plant_seeds_grow
4	38	4_used_material_form_soda
5	35	5_light_heat_pot_stove
6	32	6_offspring_birth_eggs_behavior
7	22	7_moon_month_december_changes
8	10	8_bears_polar_bear_black

TABELA 7.5: 10 najczęstszych słów kluczowych i liczbą pytań, w których występują w grupie pytań odstających

Słowo klucz	Liczba pytań
desert	5
rabbit	4
dog	8
live	15
water	21
example	13
place	10
planet	5
wind	3
money	3

7.4 HotpotQA

HotpotQA [YQZ⁺18] jest benchmarkiem stworzonym na potrzeby problemu odpowiadania na wielodziedzinowe pytania otwarte, w którym autorzy skupili się na wymaganiu dotyczącym wnioskowania po kilku dokumentach. Nie ogranicza to testowanego rozwiązania jedynie do zewnętrznych źródeł informacji, a poprzez specjalnie skonstruowane pytania, zmusza do połączenia faktów z więcej niż jednego tekstu. Dzieje się to m.in. dzięki formatowi zadań, który opiera się najczęściej na porównaniu informacji o dwóch instancjach w ramach przedstawionego zadania. Twórcy stworzyli zbiór 112,779 pytań w języku angielskim podzielonych na część treningową, walidacyjną oraz testową. Przykładowe pytania wraz z oczekiwanaą odpowiedzią zostały umieszczone w tabeli 7.6. Dodatkowo w ramach benchmarku podano bazę ponad 5 mln dokumentów będących akapitami

tekstu w hasłach z Wikipedii. Do ewaluacji oryginalnie stosowane jest dokładne dopasowanie oraz miara F1, przy czym punktem odniesienia są kolejno 23.95 oraz 32.89.

TABELA 7.6: Przykładowe pytania z benchmarku HotpotQA

Pytanie	Oczekiwana odpowiedź
Which magazine was started first Arthur's Magazine or First for Women?	Arthur's Magazine
The Dutch-Belgian television series that "House of Anubis" was based on first aired in what year?	2006
Are both The New Pornographers and Kings of Leon American rock bands?	no
Fast Cars, Danger, Fire and Knives includes guest appearances from which hip hop record executive?	Jaime Meline

7.4.1 Analiza dokumentów

Baza dokumentów udostępnionych przez autorów benchmarku HotpotQA składa się z artykułów pobranych z internetowej encyklopedii Wikipedia opisanej w podrozdziale 7.5. Z uwagi na ogromną różnorodność treści oraz rozmiar danych zdecydowaliśmy się nie przeprowadzić analizy tematycznej. Chcemy jednak zauważyć, iż zebrane dokumenty pochodzą z anglojęzycznej wersji internetowej encyklopedii sprzed października 2017 roku.

7.4.2 Analiza pytań

W wyniku przeprowadzonej za pomocą BERTopic analizy tematycznej pytań ze zbioru walidacyjnego powstało 95 różnych tematów. Pierwsze 25 najliczniejszych grup tematycznych (w tym elementy odstające oznaczone numerem -1) zamieściliśmy w tabeli 7.7. Zaobserwować można dużą różnorodność, co potwierdza, że jest to benchmark składający się z pytań pochodzących z wielu dziedzin.

Warto zaznaczyć, że w trakcie analizy zauważliśmy, iż HotpotQA zawiera pytania dotyczące tematów, które mogą zostać uznane przez niektóre instytucje za nieodpowiednie. Należy mieć to na uwadze, gdyż tak sklasyfikowane dane wejściowe dla udostępnionych API modeli językowych są często odrzucane. Przykładem są znalezione przez nas pytania dotyczące Adolfa Hitlera (6 pytań zawierających słowo "Hitler") lub partii nazistowskiej (8 pytań zawierających słowo "nazi").

TABELA 7.7: 25 najbardziej licznych grup tematycznych pytań benchmarku HotpotQA otrzymanych za pomocą BERTopic.

Temat	Liczba pytań	Nazwa Tematu
-1	2410	-1_film_did_actor_american
0	856	0_album_band_song_singer
1	215	1_novel_author_novelist_published
2	203	2_university_school_college_high
3	194	3_footballer_football_club_league
4	140	4_king_princess_father_mother
5	132	5_company_headquartered_ceo_corporation
6	131	6_born_earlier_arthur_drew
7	125	7_species_genus_plants_plant
8	115	8_magazine_published_magazines_monthly
9	106	9_actress_role_starred_series
10	98	10_canal_river_lake_bridge
11	86	11_profession_common_scope_share
12	78	12_game_video_board_games
13	77	13_composer_composed_music_classical
14	74	14_directors_director_filmaker_film
15	64	15_physicist_physics_nobel_prize
16	62	16_korean_south_kim_film
17	58	17_olympic_medal_gold_olympics
18	57	18_directed_film_movie_starring
19	57	19_population_census_2010_county
20	56	20_district_cities_china_province
21	54	21_election_republican_politician_senator
22	53	22_driver_racing_prix_race
23	52	23_tennis_sport_slam_players

7.5 Wikipedia

Wikipedia [Wik22b] jest internetową, wielojęzyczną encyklopedią, która stanowi bazę artykułów dotyczących przeróżnych definicji. Jej redaktorami są użytkownicy, którzy mają możliwość tworzenia zarówno nowych wpisów, jak i aktualizowania istniejących informacji. Skupiając się na anglojęzycznej części, dostępne jest ponad 6.5 miliona dokumentów, które można swobodnie przeglądać, a dzięki udostępnionym funkcjom i API istnieje także możliwość ich filtrowania i sortowania m.in. po zgodności z tematem. Obecnie Wikipedia stanowi jedno z największych i zarówno wiarygodnych źródeł informacji dostępnych w internecie. W naszej pracy korzystamy ze wspomnianego API przy wyszukiwaniu informacji do otwartego systemu.

Rozdział 8

Proponowany system

Pracę nad budową systemu, nazwanego przez nas PIQARD (prompted Intelligent Question Answering with Retrieval of Documents), odpowiadającego na pytania w wielu dziedzinach z wyszukanym kontekstem w dokumentach, którego częściami składowymi jest biblioteka umożliwiająca jego stworzenie z komponentów, a także aplikacja internetowa udostępniająca interfejs użytkownika z gotową konfiguracją, rozpoczęliśmy od określenia struktury oraz schematu architektonicznego.

Poniżej przedstawiamy metodykę pracy, opis zastosowanych rozwiązań oraz szczegółowe implementacyjne stworzonego projektu.

8.1 Metodyka pracy

Prace koncepcyjne oraz implementacyjne odbywały się w metodyce zwinnej zwanej Scrum. Pozwoliła ona kontrolować i nadzorować przyrostowe budowanie systemu poprzez ustalone spotkania zespołu autorów nazywane sprintami. W trakcie ich trwania planowaliśmy kolejne zadania, a także tworzyliśmy retrospekcje poprzednich zadań. Wyznaczyliśmy również osobę odpowiedzialną za przestrzeganie reguł i praktyk Scrum, której rolę określa się jako Scrum Master. Dodatkowo podczas tworzenia proponowanego rozwiązania przyświecały nam cztery wartości manifestu metodyki zwinnej, z których szczególną uwagę poświęcaliśmy stwierdzeniom "ludzie oraz interakcje ponad procesami i narzędziami" oraz "reagowanie na zmianę ponad trzymanie się planu".

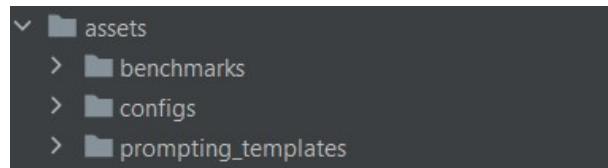
8.2 Struktura projektu

Stworzony projekt składa się z sześciu głównych folderów:

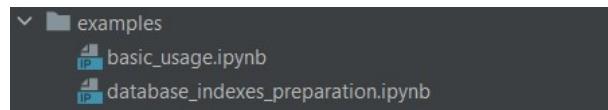
- **assets** - zawiera zasoby umożliwiające wykorzystanie komponentów m.in. pliki konfiguracji systemu, bazy dokumentów, bazy pytań z benchmarków oraz pliki tekstowe z promptami (rys. 8.1)
- **examples** - zawiera pliki python jupyter notebook .ipynb, wraz z przykładami wykorzystania biblioteki oraz skryptami umożliwiającymi stworzenie indeksów dla wybranych baz dokumentów (rys. 8.2)
- **experiments** - zawiera pliki służące do przetestowania wybranych strategii promptingu oraz odpowiadających im promptów (rys. 8.3)
- **piqard** - zawiera moduły biblioteki służącej do tworzenia systemu odpowiadającego na pytania w wielu dziedzinach z wyszukiwaniem kontekstu w dokumentach (rys. 8.4, 8.5, 8.6, 8.7)

- **piqard_api** - zawiera plik z możliwością uruchomienia lokalnego serwera udostępniającego stworzony system (rys. 8.8)
- **piqard_app** - zawiera pliki tworzące aplikację internetową z interfejsem użytkownika (rys. 8.9)

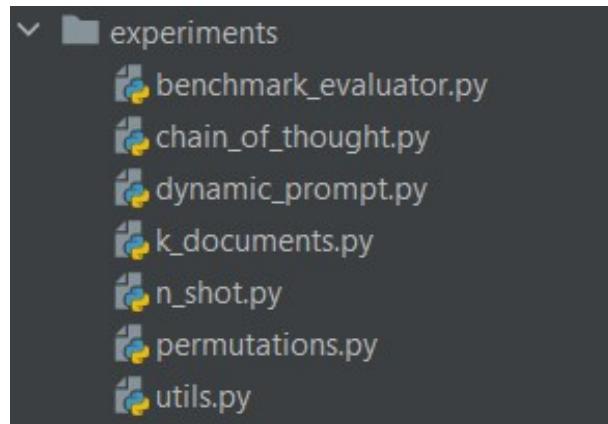
W następnych rozdziałach opiszemy dokładnie architekturę proponowanego systemu oraz szczegółowe implementacyjne stosowanych rozwiązań w bibliotece i aplikacji internetowej.



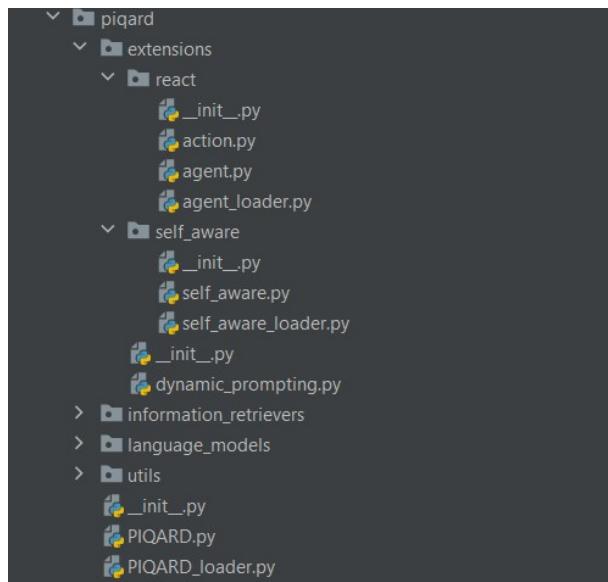
RYSUNEK 8.1: Struktura projektu — foldery zasobów



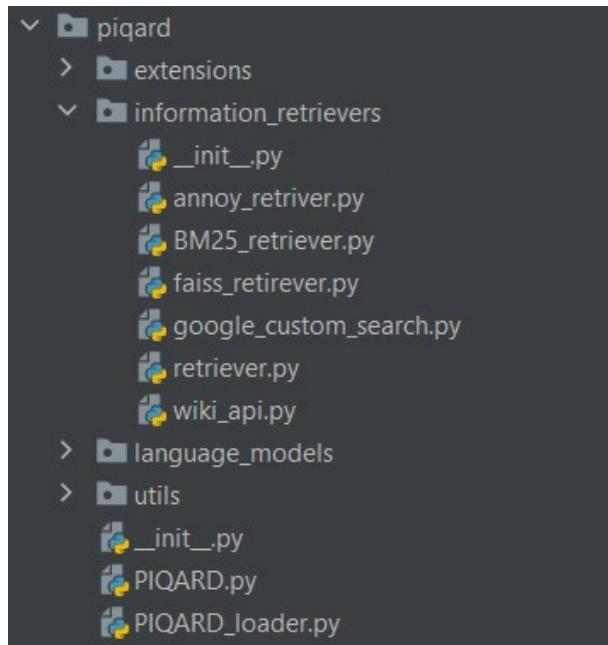
RYSUNEK 8.2: Struktura projektu — przykłady użycia biblioteki



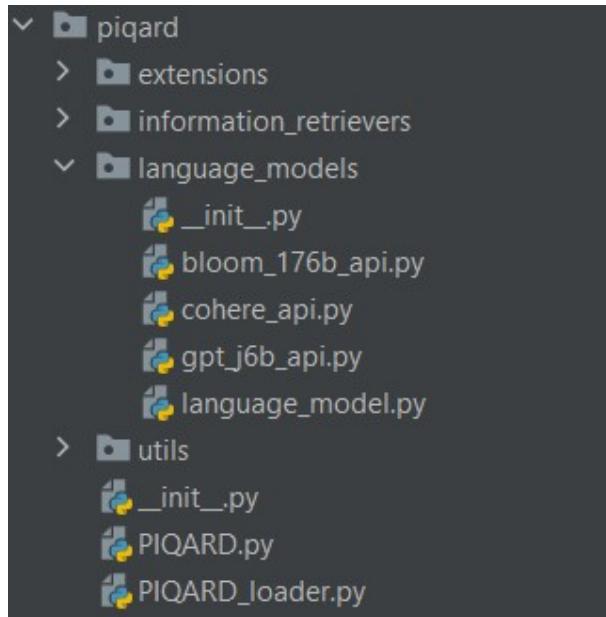
RYSUNEK 8.3: Struktura projektu - eksperymenty



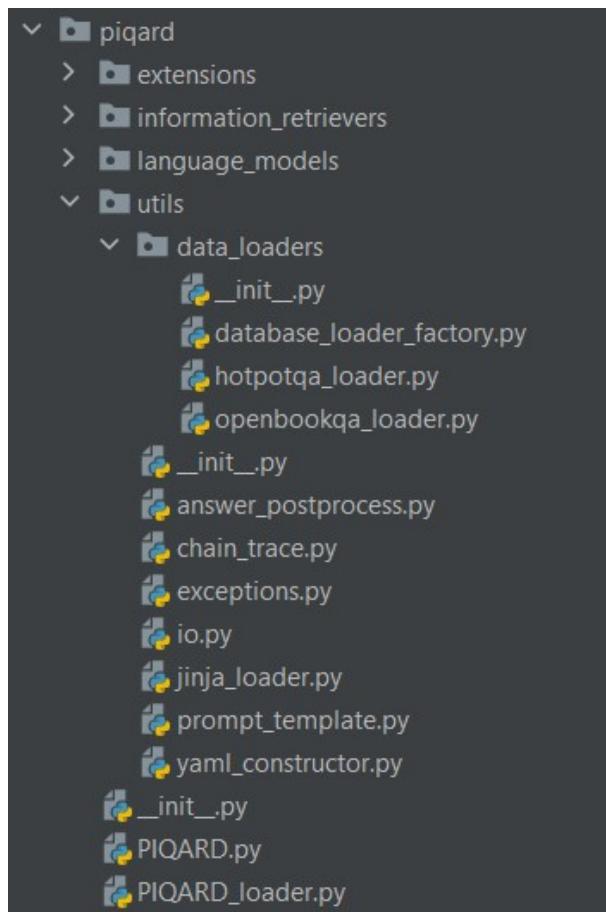
RYSUNEK 8.4: Struktura projektu — moduł biblioteki extensions



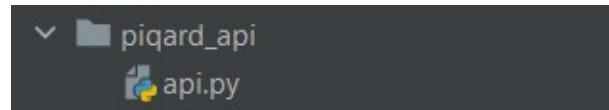
RYSUNEK 8.5: Struktura projektu — moduł biblioteki information_retrievers



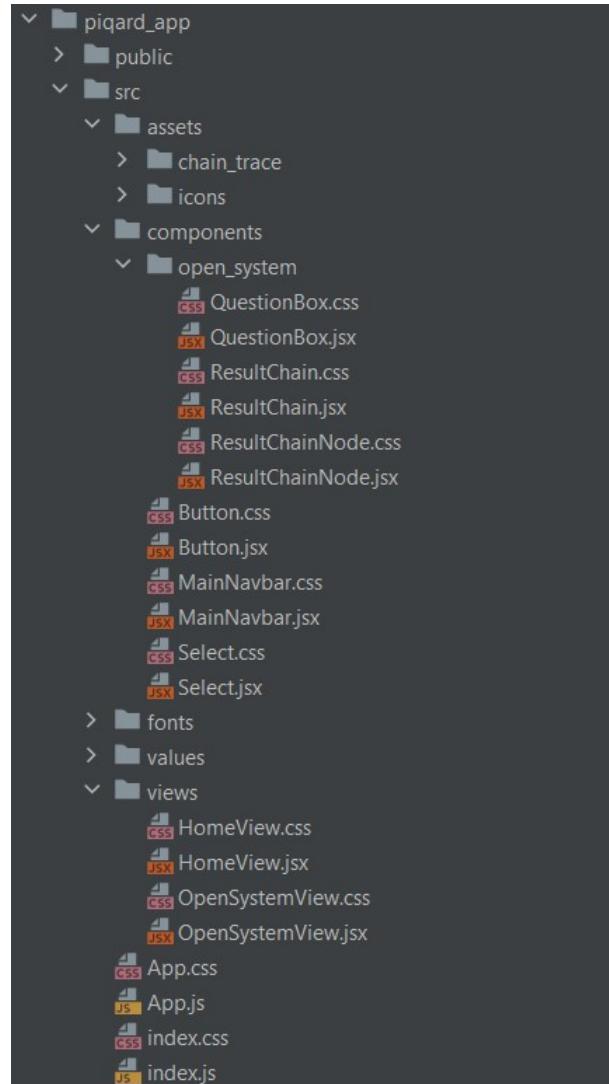
RYSUNEK 8.6: Struktura projektu — moduł biblioteki language_models



RYSUNEK 8.7: Struktura projektu — moduł biblioteki utils



RYSUNEK 8.8: Struktura projektu — pliki api



RYSUNEK 8.9: Struktura projektu — aplikacja internetowa z interfejsem użytkownika

8.3 Architektura

Tworzone rozwiązanie zaprojektowaliśmy z myślą o zapewnieniu założeń i wymagań przedstawionych w celach pracy. Jednym z nich było osiągnięcie łatwej w utrzymaniu, modułowej biblioteki, w związku z czym zastosowaliśmy wzorce projektowe, które pozwalają utrzymać czystość kodu oraz powtarzalność zastosowania zbudowanych komponentów. Dodatkowo wzorce architektoniczne mają swoją rolę w wydzieleniu w prezentowanej pracy osobnego modułu odpowiadającego za aplikację internetową z interfejsem użytkownika.

Model klient-serwer (rys. 8.10) pozwolił na podział zadań między dwa moduły. Aplikacja *piqard_app* przyjęła obowiązki zapewnienia użytkownikowi wizualnego dostępu do skonfigurowanego systemu. Jako klient prezentuje ona otrzymane wyniki odpowiedzi na pytania poprzez interakcje z użytkownikiem, a następnie wysyłanie żądania do serwera. Rolę serwera przyjęło stworzone API *piqard_api*, które działa w trybie pasywnym i oczekuje na żądanie ze strony klienta. Udostępnia on usługę odpowiedzi na pytanie, dzięki skonfigurowanemu rozwiązaniu stworzonemu z komponentów biblioteki.

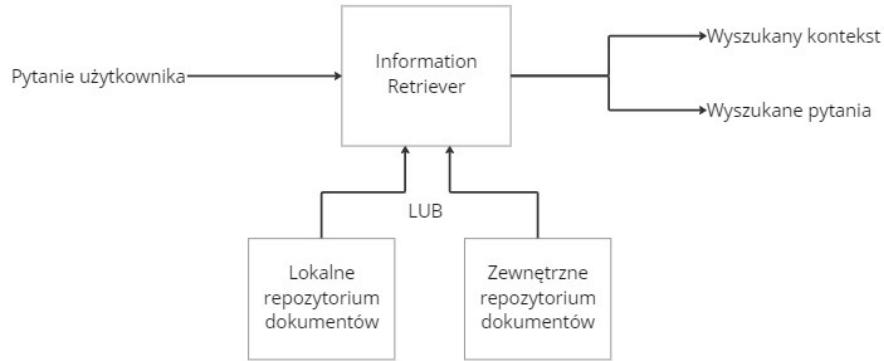


RYSUNEK 8.10: Model klient serwer, zastosowany w podziale ról interakcji z użytkownikiem oraz generowaniu odpowiedzi na pytanie wykorzystując skonfigurowane rozwiązanie.

Biblioteka *piqard* powstała stosując paradygmat programowania obiektowego, gdzie poszczególne składowe programu traktowane są jako obiekty i komunikują się ze sobą w celu wykonania zadania. Podzielona jest na trzy główne części, z których każda odpowiada za inną funkcję i może być wykorzystywana jako osobny, niezależny element. Zaprojektowane są w konwencji behawioralnego wzorca projektowego zwanego strategią, który pozwala na oddzielną implementację indywidualnych algorytmów rozwiązujących ten sam problem na różne sposoby. Docelowa klasa obsługująca konfigurację systemu posiada jedynie pole służące przechowaniu aktualnie wybranej strategii, co jest możliwe dzięki implementacji wspólnego interfejsu dla danej grupy klas. Staje się ona wtedy niezależna od stosowanego sposobu, co pozwala na modyfikowanie i rozszerzanie algorytmów bez ingerencji w jej kod.

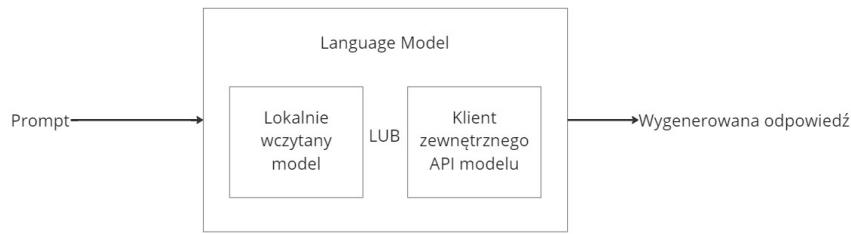
Moduł *information_retrevers* (rys. 8.11) udostępnia obiekty zawierające metody wyszukiwania informacji. Każdy z obiektów implementuje interfejs, który pozwala na odnalezienie dokumentów z odpowiedniego źródła. Możliwe jest także zastosowanie niektórych z nich do wyszukiwania podobnych do użytkownika zapytań, po uprzednim przygotowaniu takiego zabioru.

Moduł *language_models* (rys. 8.12) umożliwia wykorzystanie obiektów, których podstawą są modele języka naturalnego oparte na sztucznych sieciach neuronowych. Implementują one interfejs z metodą generującą tekst będący w domyśle odpowiedzią na zadane pytanie. Możliwe jest,



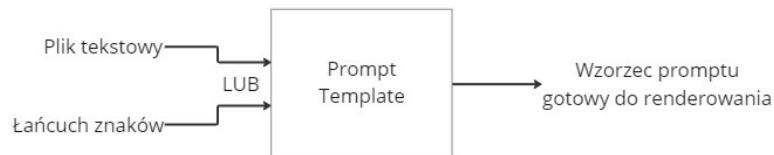
RYSUNEK 8.11: Schemat obiektu odpowiedzialnego w bibliotece za wyszukiwanie informacji.

aby dany obiekt opierał swoje działanie na modelu wczytanym lokalnie w trakcie inicjacji, bądź był klientem, który odwołuje się do niezależnego, zewnętrznego API udostępnionego przez inne biblioteki i systemy.



RYSUNEK 8.12: Schemat działania obiektu odpowiedzialnego w bibliotece za generowanie odpowiedzi.

Klasa `proprompt_template` z modułu `utils` (rys. 8.12) zapewnia możliwość załadowania przygotowanego wcześniej promptu z pliku tekstowego bądź podanego łańcucha znaków, który jest wzorcem danych wejściowych dla modelu języka. Każdy z nich napisany jest w określonym dla danej strategii promptingu stylu, jest powtarzalny i posiada specjalnie wydzielone miejsca, aby umieścić w nich pytanie użytkownika wraz z ewentualnym kontekstem i opcjonalnymi możliwościami odpowiedzi.



RYSUNEK 8.13: Schemat działania obiektu odpowiedzialnego w bibliotece za wczytanie wzorca promptu.

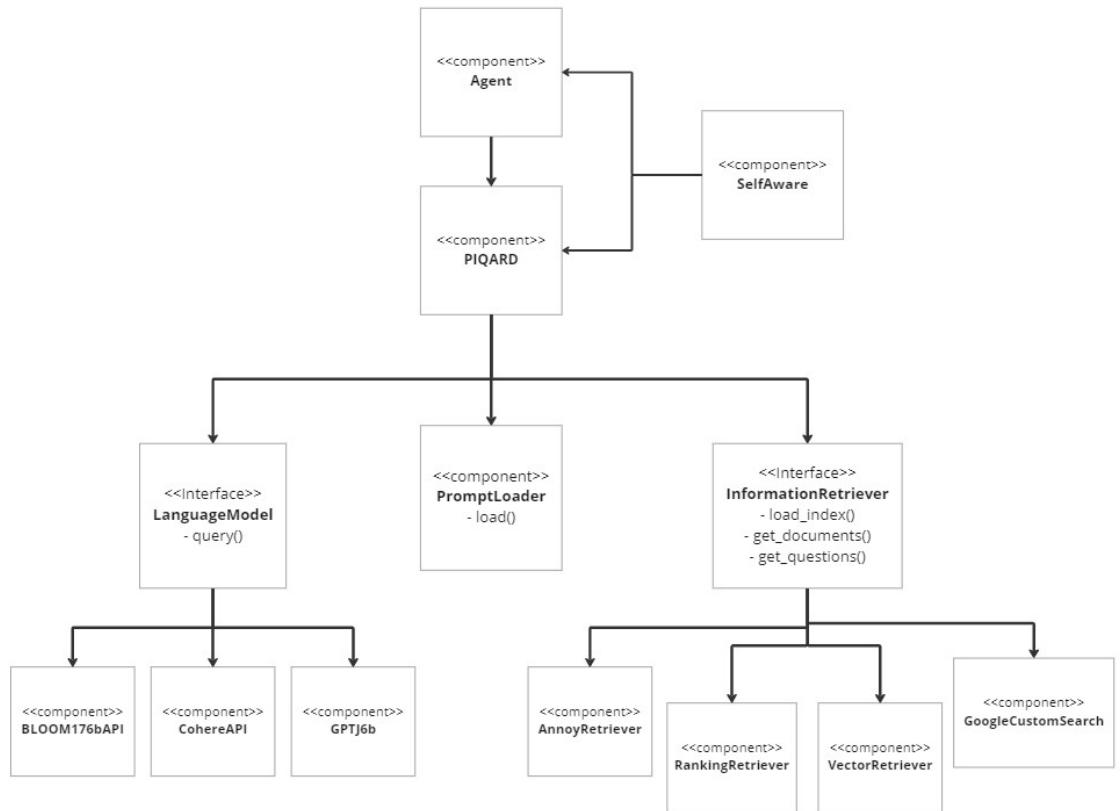
Wybierając poszczególne obiekty z powyższych modułów oraz łącząc je w odpowiedniej ko-

lejności, powstaje konfiguracja systemu odpowiadającego na pytanie użytkownika z wyszukiwaniem kontekstu w dokumentach. W celu ułatwienia z ich korzystania stworzyliśmy klasę *PIQARD*, która jest fasadą, czyli wzorcem strukturalnym stanowiącym prosty interfejs dla złożonego podsystemu. Ukrywa ona funkcjonalność składowych obiektów przed resztą kodu i udostępnia metodę, której zadaniem jest zwrócenie odpowiedzi, abstrahując od wewnętrznego działania pomniejszych elementów. Kosztem ograniczenia możliwości modyfikacji wewnętrznego zachowania użytkownik uzyskuje sposób na prostą integrację własnej aplikacji z proponowanym przez nas rozwiązaniem.

Możliwość konfiguracji w przypadku prezentowanej biblioteki oraz stworzonej fasady wiąże się z wyborem konkretnych obiektów, czyli rozwiązań kolejnych etapów udzielenia odpowiedzi na pytanie. Inicjalizacja klasy *PIQARD* wymaga podania instancji obiektu z modułu wyszukiwania informacji oraz instancji z modułu generowania odpowiedzi. To one składają się na bazowe elementy przetwarzające dane w systemie. Inicjalizację można dokonać podając manualnie obiekty do konstruktora, jednak dodatkowo przygotowaliśmy klasę *PIQARDLoader*, która pozwala skonstruować system, podając jego konfigurację w pliku.

W celu rozdzielenia tworzonych komponentów od możliwości ładowania ich za pomocą pliku wykorzystaliśmy strukturalny wzorzec projektowy zwany dekoratorem, który pozwala dodać klasie nowe obowiązki poprzez opakowanie jej w nowy obiekt posiadający dane metody. Nie ingeruje to w bezpośrednią implementację, dzięki czemu możliwe jest proste dostosowanie ewentualnych nowych modułów.

Dodatkowo, aby zastosować wybrane strategie promptingu oparte nie tylko na wzorcu promptu, ale również na schemacie działania, przygotowaliśmy moduł *extensions* z klasą *Agent* dla metody ReAct oraz klasą *SelfAware* dla metody z dodatkowym pytaniem o potrzebę wyszukiwania kontekstu. Klasa *Agent* dziedziczy z klasy *PIQARD* i jest jej rozszerzeniem pozwalającym na wykorzystanie możliwości bazowego systemu wraz z zastosowaniem zaimplementowanych akcji w formie wzorca behawioralnego strategia. Posiada także analogiczną klasę konstrującą system poprzez inicjalizację obiektu z pliku konfiguracyjnego, zwaną *AgentLoader*. Natomiast klasa *SelfAware* działa w oparciu o konwencję strukturalnego wzorca projektowego zwanego pełnomocnikiem. Można go zastosować, gdy istnieje potrzeba dodania funkcjonalności przed lub po wykorzystaniu bazowego obiektu. W przypadku tej klasy inicjalizowane są dwa obiekty będące instancjami klasy *PIQARD* bądź dziedziczące po niej, lecz przed wywołaniem jednego z nich dokonywany jest konkretny wybór. W jej przypadku przygotowaliśmy także analogiczną do poprzedników klasę konstrującą system z pliku nazwaną *SelfAwareLoader*.



RYSUNEK 8.14: Schemat zależności obiektów wykorzystywanych w bibliotece.

8.4 Implementacja

Biblioteka składająca się z komponentów i funkcji pozwalających konstruować system odpowiadający na pytania użytkownika z wyszukiwaniem kontekstu w dokumentach została zaimplementowana w języku Python w wersji 3.9., która posiada stabilne wydanie oraz zapewnione wsparcie twórców. Wybrałyśmy tę technologię z uwagi na prostą składnię kodu, która zwiększa jego czytelność oraz pozwala na zrozumienie działania nawet przez początkujących programistów. Umożliwia ona także szybki rozwój systemu i zapewnia ogromną społeczność aktywnie udzielających się specjalistów. Dodatkowo język Python jest często wykorzystywany przez inżynierów sztucznej inteligencji w implementowaniu nowych rozwiązań i przeprowadzaniu eksperymentów. Jego popularność w tej dziedzinie wynika m.in. z dostępności wielu bibliotek wspomagających operacje numeryczne, pomagających w analizie danych oraz dostarczających gotowe podstawowe i zaawansowane metody uczenia maszynowego. Przykładem są numpy, pandas, a także Hugging Face transformers, które zapewniły nam niektóre algorytmy i modele bez konieczności ich ponownej implementacji. Język Python posiada również swoje wady, z których wymienić można m.in. słabą wydajność pamięciową oraz wysoką złożoność czasową w porównaniu do innych języków jak np. C++. Zważając jednak na pozytywne aspekty, jakie niesie za sobą wykorzystanie tej technologii oraz biorąc pod uwagę inne rozwiązania, ostatecznie zdecydowaliśmy się właśnie na nią.

Zapewniliśmy utrzymanie jakości tworzonego rozwiązania oraz czystości kodu, poprzez stosowanie dobrych praktyk m.in. DRY (Don't Repeat Yourself) oraz stosowanie się do konwencji formatowania black opartego na standardzie PEP8. Dodatkowo do testowania poprawność działania poszczególnych komponentów użyliśmy biblioteki PyTest oraz napisaliśmy dokumentację komponentów w formacie Sphinx.

Interfejs użytkownika w formie aplikacji internetowej zaimplementowaliśmy w języku javascript korzystając z biblioteki React.js. Zapewnia ona szybkość działania po stronie klienta, dowolność w wyborze architektury oraz łatwość integracji z innymi rozwiązaniami m.in. stworzonego przez nas API w bibliotece języka Python uicorn. Dodatkowo React.js wykorzystywany jest przez duże korporacje, jak np. Meta, dzięki czemu zapewniono długoterminowe wsparcie programistów.

Całość kodu, czyli bibliotekę wraz z aplikacją internetową, udostępniliśmy w postaci open-source dostępnej do pobrania na platformie github korzystając z linku: <https://github.com/plaskod/piqard>. W dalszej części pracy przedstawiamy wybrane szczegóły dotyczące zaimplementowanych modułów biblioteki oraz aplikacji z interfejsem użytkownika.

8.4.1 Moduł information_retrievers

Moduł *information_retrievers* zawiera klasy odpowiadające za wyszukiwanie informacji zarówno w lokalnych, jak i zdalnych zbiorach dokumentów. Jego centralnym elementem jest stworzony interfejs o nazwie *Retriever*, który w swoim konstruktorze przyjmuje parametry k i n odpowiadające kolejno liczbę zwracanych dokumentów oraz liczbę zwracanych pytań w trakcie przeszukiwania baz danych. Opcjonalnie można także podać ich nazwy oraz ścieżki prowadzące do plików z dokumentami oraz pytaniami w przypadku, gdy implementowana metoda korzysta z lokalnego repozytorium. Dodatkowo interfejs Retriever udostępnia podstawowe metody wirtualne *get_documents()*, *get_questions()* oraz *load_index()*. Pierwsza z nich odpowiada za zwrócenie k wybranych dokumentów wyszukanych w zbiorze dokumentów, druga zwraca n najbardziej podobnych pytań wyszukanych w zbiorze pytań, natomiast trzecia metoda wczytuje indeks znajdujący się w podanej wcześniej ścieżce podczas inicjalizacji obiektu. Poniżej opisane klasy umożliwiają wykorzystanie wybranych metod i algorytmów wyszukiwania informacji, bazując na implementacji przedstawionego interfejsu.

BM25Retriever

Klasa *BM25Retriever* korzysta z biblioteki fastbm25¹, która implementuje algorytm BM25 w języku python. Twórcy zapewniają, że dzięki wykonanej przez nich optymalizacji złożoność obliczeniowa nie powinna być większa niż $O(N \log(N))$. Niestety w tej bibliotece nie jest możliwe własnoręczne dobranie parametrów k_1 i b opisanych w rozdziale 4.1.1, gdyż są one na stałe ustalone na następujące wartości: $k_1 = 1.5$, $b = 0.75$.

Korzystanie z algorytmu BM25 wymaga wstępnego przygotowania dokumentów i pytań. Konieczne jest, aby sposób wstępnego przygotowania był taki sam zarówno dla aktualnie przetwarzanego pytania, jak i dokumentów lub pytań znajdujących się w podanej wcześniej bazie danych. Zaproponowany przez nas algorytm wstępniego przetwarzania korzysta ze stop listy (ang. stop words), czyli usuwa słowa o małym znaczeniu w identyfikacji dokumentu, m.in. spójniki, słowa często się powtarzające. Następnie każde słowo podlega procesowi stemmingu, czyli usunięcia końcówki fleksyjnej pozostawiając tylko tę część wyrazu, która nie powinna uczestniczyć w odmianie tego słowa.

Implementowana klasa korzysta z lokalnych repozytoriów, dlatego w celu jej użycia konieczne jest podanie ścieżek do plików, gdzie przechowywana jest baza pytań oraz opcjonalnie dokumentów, a także odpowiadające im, wcześniej stworzone indeksy. Klasa ta nadpisuje wirtualne metody (*get_documents()*, *get_questions()*) interfejsu i za pomocą wspomnianej biblioteki fastBM25 wyszukuje najbardziej dopasowane dokumenty do pytania, które zostało wstępnie przetworzone.

¹<https://github.com/zhusleep/fastbm25> - dokumentacja i źródła biblioteki fastbm25

Biblioteka nie umożliwia tworzenia indeksu. Dlatego, aby utworzyć indeks, który ma zostać wczytany do klasy *BM25Retriever* należy stosować się do instrukcji umieszczonej w naszym repozytorium w pliku o ścieżce: *piqard/examples/database_indexes_bm25_preparation.ipynb*

FaissRetriever

Klasa *FaissRetriever* wykorzystuje strukturę indeksu dostarczaną z biblioteki FAISS² powstałą dzięki pracy inżynierów z placówki Facebook AI Research. Przy wywołaniu klasy ładowamy do pamięci dwa osobne pliki znajdujące się w podanym poprzez parametr folderze bazowym. Każdy plik zawiera strukturę indeksu dla *zaenkodowanych* tekstów — jeden dla pytań oraz drugi dla dokumentów kontekstowych. Wywołując funkcję wyszukującą, jako parametr podawany jest wektor odpowiadający zadanemu przez użytkownika tekstuowi zapytania, koniecznie *enkodowany* tym samym modelem, który użyty został przy tworzeniu indeksu. W analizie przeprowadzonej na zbiorze OpenBookQA, gdzie dla danego pytania wyszukiwane były fakty naukowe pomagające w udzieleniu odpowiedzi na pytanie, największą trafność spośród modeli trenowanych w technice Sentence-BERT osiągnął enkoder *multi-qa-MiniLM-L6-cos-v1*. Biblioteka Faiss zawiera selekcję typów indeksów, lecz naszym systemie zdecydowaliśmy się na typ IVFPQ, ponieważ stosuje on wszystkie wymienione w 4.2.3 optymalizacje: *indeks odwrócony*, oraz *kwantyzację wektorową*. Ten wybór pozwala nam ograniczyć objętość pamięciową struktury indeksu o kilka rzędów wielkości, który w swojej niezoptymalizowanej wersji bez kompresji wektorów wymagałby dla bazy HotpotQA zagospodarowania dodatkowych kilkunastu gigabajtów pamięci RAM. Nasz moduł nie pozwala na utworzenie indeksów, muszą one zostać wcześniej przygotowane do operowania na wybranej bazie danych i zapisane na dysku. Indeks należy stworzyć według wzoru w naszym repozytorium o ścieżce: *piqard/examples/database_indexes_faiss_preparation.ipynb*. W parametrach wymaganych do zbudowania indeksu przyjęliśmy domyślne parametry - 8 równych segmentów, na które podzielone zostały wektory w procesie *kwantyzacji wektorowej*, maksymalną ilość zwracanych sąsiadów przy jednokrotnym uruchomieniu funkcji równą 100. Wymiar danych wejściowych zależy bezpośrednio od wykorzystanego enkodera, a więc dla modelu *multi-qa-MiniLM-L6-cos-v1* jest to 384.

AnnoyRetriever

Klasa *AnnoyRetriever* korzysta z udostępnionej przez firmę Spotify biblioteki w języku C++ o nazwie Annoy³, która udostępnia interfejs także dla języka Python w bibliotece o identycznej nazwie. Przy inicjalizacji, z racji na konieczność korzystania z lokalnych repozytoriów, należy podać kilka argumentów. Po pierwsze wymagane są ścieżki do wcześniej utworzonych indeksów i baz danych, które wskazują na zawierające je pliki w utworzonym przez użytkownika folderze. Dodatkowo wymagana jest także liczba wymiarów utworzonego indeksu oraz nazwa modelu wykorzystywanego do jego enkodowania, która musi odpowiadać dostępnym w bibliotece wersjom np. *multi-qa-MiniLM-L6-cos-v1*. Ważne jest, aby wybrany model odpowiadał wariantowi użytkemu do zakodowania bazy danych, gdyż to nim enkodowane będzie pytanie użytkownika w trakcie działania systemu i wyszukiwania podobnych dokumentów.

Biblioteka nie umożliwia tworzenia indeksu. Dlatego, aby utworzyć indeks, który ma zostać wczytany do klasy *AnnoyRetriever* należy stosować się do instrukcji umieszczonej w naszym repozytorium w pliku o ścieżce: *piqard/examples/database_indexes_annoypreparation.ipynb*.

²<https://faiss.ai/> - dokumentacja i źródła biblioteki Faiss

³<https://github.com/spotify/annoy> - dokumentacja i źródła biblioteki Annoy

WikiAPI

Klasa *WikiAPI* korzysta z biblioteki o nazwie *wikipedia*, która jest interfejsem udostępnianego przez organizację MediaWiki API do języka Python. Zadaniem biblioteki jest umożliwienie wyszukiwania i dostępu do artykułów w wielojęzycznej encyklopedii internetowej o nazwie Wikipedia.

Z racji tego, że biblioteka wykorzystuje dane ze zdalnego repozytorium, nie ma potrzeby podawania ścieżek do baz danych i indeksów. Nie można też wyszukiwać pytań podobnych do aktualnie przetwarzanego pytania, ponieważ encyklopedia nie posiada takiego zbioru.

W naszej implementacji korzystamy tylko z anglojęzycznych artykułów. Klasa posiada jedną metodę *get_documents*, która zwraca k pierwszych zdań wyszukanego dokumentu. W przypadku wystąpienia wyjątków typu *PageError* (nie znaleziono strony z szukaną frazą) i *DisambiguationError* (przy wyszukiwaniu szukany ciąg znaków może odwoływać się do więcej niż jednego artykułu) zwracana jest informacja, że nie znaleziono artykułu i podawane są nazwy artykułów, w których może być zawarta informacja dotycząca szukanego wyrażenia.

GoogleCustomSearch

Klasa *GoogleCustomSearch* pozwala na wyszukiwanie informacji zawartych w artykułach stron internetowych i korzysta z dwóch bibliotek. Pierwsza z nich nazwana *request* umożliwia wysyłanie i odbieranie żądań HTTP. Dzięki niej odpytujemy Custom Search API udostępnione przez Google, które pozwala na skorzystanie z wcześniej przygotowanego silnika wyszukiwarki. Silnik wyszukiwania musi zostać wcześniej zainicjowany przez użytkownika poprzez podanie na stronie zakresu witryn internetowych do przeszukiwania, a także podstawowych opcji jak np. region. W celu użycia metody *get_documents*, która korzysta z API, należy ustawić zmienną środowiskową zawierającą identyfikator utworzonego silnika nazwaną “GOOGLE_CUSTOM_SEARCH_ENGINE_ID”, a także zmienną środowiskową odpowiadającą za przechowanie klucza użytkownika, której nazwa to “GOOGLE_CUSTOM_SEARCH_API_KEY”.

Custom Search API zwraca informacje na temat wyszukanych stron internetowych dlatego, aby wykorzystać ich treść jako dokumenty dla naszego systemu, przetwarzamy je metodą *parse_article*, która przyjmuje jako argument adres URL danego artykułu. Dzięki wykorzystaniu biblioteki *newspaper* zwracana jest jego przetworzona treść do formy ciągu znaków bez znaczników HTML.

8.4.2 Moduł language_models

Klasa *LanguageModel* służy jako jednolity interfejs do komunikacji z publicznie dostępnymi API wykorzystanych przez nas modeli językowych. Dziedzicząc po niej klasy dostosowane do każdego używanego modelu. Na pojedyncze wywołanie funkcji *query* przypada wielokrotne wysyłanie zapytań do modelu, ponieważ wybrane API mogą zwrócić niekompletną odpowiedź. Treść pierwszego zapytania do serwera z modelem zawiera prompt i zadane pytanie, a w każdym kolejnym wywołaniu treść stanowi poprzednio użyty tekst wraz z dopisaną na końcu odpowiedzią zwróconą na zapytanie, co pozwala na uzupełnienie niedokończonego tekstu przez model. Ilość wysłanych zapytań musi być również z góry ograniczona, aby nie generować zbędnego tekstu. Do tego stosujemy parametr *stop_token*, czyli symbol stopu.

Zdefiniowany w używanym promptcie *stop_token* jest znakiem lub złączeniem znaków, którego model dla zadanego pytania nie będzie generował, udzielając samej odpowiedzi na pytanie. Symbolami takimi mogą być przykładowo znaki ”||” lub ”—”. Rolą symbolu *stop_token* jest oddzielenie odpowiedzi na pytania od kolejnych pytań — ze względu na to, że prompt w formacie few-shot zawiera sekwencję naprzemiennych pytań i odpowiedzi, model zapytany o kolejne tokeny po za-

kończeniu udzielania odpowiedzi halucynuje własne pytania. Dzięki umieszczeniu tokenu stopu w promptcie pomiędzy blokami pytanie-odpowiedź, model języka wzorując się na tekście promptu, zakończy pełną odpowiedź, generując znak stopu. Gdy w odpowiedzi z API znajdziemy *stop_token*, oznacza to, że model zakończył odpowiadanie na pytanie, a więc zaprzestajemy wysyłać dalsze zapytania. Z otrzymanego tekstu odcinamy też fragment od wygenerowanego znaku do końca, ponieważ model nie musi zakończyć generację na tokenie stop i nadal może zawierać dodatkowo wygenerowany tekst. W przypadku braku wygenerowania znaku stopu przez model odpytywanie API kończymy po wykonaniu pięciu zapytań.

CohereAPI

Klasa *CohereAPI* obsługuje interfejs modelu języka dostarczanego przez organizację *Cohere*⁴. Inicjalizując klasę *CohereAPI* wczytujemy ze wcześniej ustawionej zmiennej środowiskowej systemu klucz API wygenerowany na portalu, a następnie tworzymy obiekt klienta *cohere.Client* dostarczonego przez twórców portalu w bibliotece *cohere*. Podstawowymi parametrami zapytania, są *temperature* oraz *top_k*, definiujące sposób wybierania tokenów z rozkładu prawdopodobieństwa w trakcie procesu generacji odpowiedzi opisanego w rozdziale 5. Temperaturę ustawiamy domyślnie na 0, zaś *k* najlepszych tokenów, z których próbkiwana jest odpowiedź, na 1.

API modelu Cohere pozwala na szeroką personalizację zapytania generacji tekstu i dostosowanie go do oczekiwanych wyników ponad wymienione standardowe parametry. Do wyboru są dwa modele różniące się przede wszystkim liczbą wag połączeń modelu: używany przez nas domyślny *xlarge* oraz *medium*. Interfejs oferuje szeroki wybór dodatkowych parametrów pozwalających między innymi na wybranie maksymalnej ilości zwróconych tokenów lub zdefiniowanie symbolu stopu. Celem unifikacji interfejsu naszej biblioteki, korzystamy jednak tylko z trzech dodatkowych parametrów. Pierwszy z nich to *max_tokens*, czyli maksymalna ilość zwracanych tokenów przez model, ustawiona na 50. Kolejny parametr *truncate* definiuje sposób odcięcia tekstu w przypadku gdy tekst wejściowy podany przez nas w zapytaniu przekracza 2048 tokenów. Parametr ten ustawiony został na wartość *START*, a więc korzystając ze zbyt długiego promptu w formacie few-shot odcięte zostaną jedynie początkowe instrukcje, a nie właściwe pytanie zadane przez użytkownika, umieszczone na końcu tekstu.

BLOOM176bAPI

Twórcami modelu BLOOM⁵ są członkowie grupy badawczej BigScience Research Workshop. Autorzy udostępnili API modelu w kilku wersjach na repozytorium Huggingface. Skorzystaliśmy z największego dostępnego z nich, a więc o 176 miliardach parametrów. Podobnie jak w klasie *CohereAPI*, w trakcie inicjalizacji wczytywany jest klucz API ze stworzonej wcześniej zmiennej środowiskowej, a następnie ustawiane są parametry do zapytań — domyślnie temperaturę na wartość 10^{-6} (zblizona do 0), oraz *k* najlepszych wybranych tokenów na 1. W przeciwnieństwie do Cohere API BLOOM API nie skróci wiadomości w przypadku przekroczenia maksymalnej ilości tokenów, która wynosi 1000 dla opisywanego interfejsu, lecz zwróci błąd. Z tego powodu stosujemy preprocessing tekstu przed wysłaniem zapytania, który pozbędzie się początku promptu, jeżeli długość wiadomości zostanie przekroczona. Do określenia ilości tokenów używamy udostępnionego przez BigScience Workshop tokenizera *BloomTokenizerFast*, znajdującego się w bibliotece *transformers* udostępniającej zasoby z platformy Hugging Face.

⁴<https://cohere.ai/> - Strona firmy Cohere zawierająca API modelu języka wraz z dokumentacją

⁵<https://huggingface.co/bigscience/bloom> - Strona modelu BLOOM na repozytorium Hugging Face zawierająca API do komunikacji z modelem

GPTj6bAPI

API modelu GPT-J⁶ o liczbie 6 miliardów parametrów zostało również udostępnione na repozytorium Hugging Face. W trakcie inicjalizacji wczytujemy klucz API ze stworzonej wcześniej zmiennej środowiskowej, a następnie odpytujemy interfejs bez użycia dodatkowych parametrów.

8.4.3 Data_loaders

Zaimplementowana została biblioteka *Data_loaders*, która służy do wczytania wybranego zbioru dokumentów oraz pytań. Znajduję się w niej klasa *DatabaseLoaderFactory*, która jest interfejsem zawierającym dwie funkcje *load_documents* oraz *load_questions*. W celu poprawnego wczytania użycia biblioteki należy przygotować plik o nazwie *{nazwa_bazy_danych}.loader.py*. Plik ten musi implementować dwie wspomniane wcześniej funkcje i przygotować dane w taki sposób, aby zwarcane dokumenty były listą zawierającą treść dokumentów jako ciąg znaków, a pytania były listą zawierającą słowników z następującymi kluczami: *id*, *text*, *possible_answers*, *answer*. W ramach naszego systemu zostały przygotowane dwa pliki *hotpotqa_loader* i *openbookqa_loader*, które odpowiednio wczytują bazy danych benchmarków HotpotQA i OpenbookQA

8.4.4 Jinja2

W celu szybkiego prototypowania aplikacji bazujących na modułach *language_models* oraz eksperymentowania z wieloma wariantami promptów zdecydowaliśmy się wykorzystać silnik szablonów *Jinja2*. Jego zaletą jest możliwość zagnieźdzenia elementów języka programowania takich jak instrukcje warunkowe lub pętle w plikach tekstowych. Jest to rozwiązanie głównie wykorzystywane przez programistów aplikacji internetowych w bibliotece języka Python *Django* oraz *Flask* przy tworzeniu dynamicznych stron internetowych do renderowania danych. W naszym systemie sprawdziło się to w roli generowania dynamicznych promptów (przykład wzorca i składni w A.3) oraz wzorców ze zmienną liczbą przykładów lub dokumentów wskazanych jako kontekst. Pozwoliło to również na oddelegowanie zadań takich jak iteracyjne wstawianie tekstu do promptu, co w konsekwencji upraszcza logikę kodu w języku Python.

JINJALoader

Klasa *JINJALoader* wczytuje szablon za pomocą statycznej metody *load*, tworzącinstancję środowiska *jinja2.Environment* przechowującą zmienne, metadane i współdzielone zasoby silnika szablonów, a następnie wczytując plik tekstowy z podanej ścieżki jako argument *template* przetwarzający metodą *from_string*. Ostatecznie zwracany jest obiekt typu *jinja2.Template* będący szablonem, do którego można wstawić dane tekstowe, gdy jest to potrzebne w kolejnych krokach przetwarzania.

8.4.5 promptTemplate

Klasa *promptTemplate* została stworzona do obsługi promptów, wczytania ich przy inicjalizacji systemu oraz przechowywania ciągu znaków do usunięcia w odpowiedzi w trakcie przetwarzania. Konstruktor przyjmuje dwa argumenty: *template*, będący ścieżką do pliku przechowywanego lokalnie, oraz *fix_text*, domyślnie przyjmujący wartość literalu pustego. Szablon zostaje zainicjalizowany przez użycie metody *load* klasy *JINJALoader*. Do szablonu można wstawić tekst metodą *render*

⁶<https://huggingface.co/EleutherAI/gpt-j-6B> - Strona modelu GPT-J 6B na repozytorium Hugging Face zawierająca API do komunikacji z modelem

przekazujący podane argument *kwargs* do metody *render* obiektu *jinja2.Template*. Argumenty *kwargs* pochodzą ze słownika, w którym klucze to nazwy pól w szablonie, a wartości to tekst przeznaczony do zamieszczenia w danym polu. Uzupełniony danymi szablon zostaje zwrócony jako typ *str* –łańcuch znaków, który następnie jest przetwarzany metodą statyczną *promptTemplate.preprocess_template* usuwającą niepotrzebne białe znaki mogące powstać przy renderowaniu szablonu.

8.4.6 Klasa PIQARD

Klasa PIQARD łączy wszystkie komponenty systemu potrzebne do wyszukiwania informacji, generowania tekstu przez modele języka oraz obsługę przepływu danych pomiędzy modułami. Jest to interfejs wykorzystujący strukturalny wzorzec projektowy fasady — użyty, aby ukryć szczegóły funkcjonalności komponentów składających się na skomplikowany system. Pozwala to na uproszczenie tworzenia API, a z perspektywy użytkownika znacząco ułatwia interakcję z systemem w przypadku zaimportowania tej klasy z biblioteki. Korzystanie z *PIQARD* wymaga podania dwóch komponentów: strategii promptingu zapisanej w obiekcie *promptTemplate*, modelu języka *LanguageModel* wykonującą wybraną strategię, a opcjonalnie obiektu typu *Retriever* do wyszukiwania informacji. Tak skonstruowana instancja klasy *PIQARD* stanowi funkcjonalny system udzielania odpowiedzi na pytania, z którego można korzystać wywołując go jak funkcję, a pytanie podając jako argument *query*. Logika przetwarzania po wywołaniu systemu jest następująca:

1. Sprawdzane jest, czy obiekt obsługujący wyszukiwanie informacji został podany jako komponent system udzielania odpowiedzi na pytania, a jeżeli tak, to pozyskiwane są dokumenty, wywołując metodę *get_documents*. Pozyskiwane są również przykłady podobne do *query* w przypadku, gdy w obiekcie *Retriever* ustalono tryb wyszukiwania podobnych pytań, np. w realizacji strategii promptingu dynamicznego.
2. Konstruowany jest prompt integrujący ze sobą pytanie w *query* oraz wyszukane dokumenty jako kontekst z określoną strategią promptingu, renderując je przez metodę *render* obiektu *promptTemplate*. Mając skompilowany prompt staje się on parametrem inicjalizacji instancji klasy *ChainTrace* odpowiedzialnej za przechowywanie informacji o każdym z etapów realizacji strategii promptingu przez model języka oraz wyszukanych dokumentów.
3. Wywoływana jest metoda *query* interfejsu modelu języka podając mu prompt jako argument. W konsekwencji bezbłędnego odpytania modelu otrzymywany jest wygenerowany tekst zapisywany w obiekcie *ChainTrace* jako węzeł. Węzeł oznacza znacznik *thought* w przypadku gdy kontynuacja sekwencji jest krokiem pośrednim realizowania wieloetapowej strategii, natomiast znacznik *finish* oznacza zakończenie.
4. Wygenerowany tekst, będący konkatenacją tekstu na wejściu w promptcie i kontynuacji sekwencji wygenerowanej przez model języka, jest poddawany przetwarzaniu końcowemu wyodrębniającemu ostateczną odpowiedź.

Wynikiem wywołania *PIQARD* jest słownik zawierający ustrukturyzowaną informację o krokach przetwarzania i odpowiedzi na zadane pytanie. Warto zauważyć, że poprzez dobranie komponentów umożliwia ona dostosowanie systemu odpowiadającego na zadane pytanie do własnych potrzeb.

8.4.7 ChainTrace

Klasa *ChainTrace* jest odpowiedzialna za zapisywanie przepływu danych w systemie udzielania odpowiedzi na pytania. Początkowo zaimplementowana została na potrzeby strategii ReAct, ale

ostatecznie stała się kluczowym elementem każdego przetwarzania, niezależnie od wariantu skonfigurowanego systemy w klasie *PIQARD*. *ChainTrace* realizuje wzorzec behawioralny łańcucha zobowiązań w celu przechowywania informacji o danym etapie wykonywania strategii promptingu i interakcji modelu języka z zewnętrznymi źródłami informacji. Dodatkowym zadaniem tej klasy jest wskazanie momentu, w którym model języka udzielił ostatecznej odpowiedzi, kończąc tym działanie systemu. Każdorazowo, gdy klasa *PIQARD* jest wywoływana, tworzony jest nowy zapis, a interakcje między użytkownikiem a systemem są dodawane do tego zapisu metodą *add*. Interakcje są dodawane jako nowe węzły w łańcuchu będąc kolejną instancją obiektu *ChainTrace* o zdefiniowanym typie węzła jako jedno z: *base-prompt*, *thought*, *action*, *observation* lub *finish*. Klasa *ChainTrace* zapewnia metody do dodawania nowych węzłów do zapisu, jak również metody do uzyskiwania dostępu do zapisu. Przykładami takich metod są *compose*, która zwraca cały zapis jako ciąg znaków, oraz *get_deepest_node*, zwracająca ostatni węzeł w zapisie. Funkcjonalność klasy może być wykorzystana do generowania historii, która będzie przedstawiona użytkownikowi jako wynik przetwarzania. Może to być szczególnie przydatne dla skomplikowanych strategii promptingu, które obejmują wiele etapów generowania tekstu i wyszukiwania informacji.

8.4.8 Rozszerzenia biblioteki

oprócz dostarczania możliwości personalizacji i dopasowania odpowiednich komponentów budujących system udzielania odpowiedzi na pytania użytkownika biblioteka posiada rozszerzenia w postaci zaimplementowanych strategii promptingu do realizacji złożonych interakcji z modelem języka.

Prompting dynamiczny

Funkcją wspierającą strategię promptingu dynamicznego jest *get_prompt_examples*. Jako argumenty przyjmuje ona pytanie (w domyśle przypadek testowy) orazinstancję obiektu Retriever wyszukującego podobne przykłady do pytania przy wywołaniu metody *get_questions*. Dla każdego znalezionego przykładu wyszukiwane są podobne dokumenty, korzystając z metody *get_documents*. Zwracany przez funkcję jest słownik przechowujący informację o przykładach i ich powiązanych dokumentach potrzebnych do wypełnienia danymi szablonu *promptTemplate* o promptcie wskazanym w A.3 na stronie 112.

Strategia SelfAware

Klasa *SelfAware* odpowiada za realizację strategii promptingu odpytując model języka czy zadane pytanie wymaga dostępu do zewnętrznego źródła informacji. Domyślnym wzorcem jest prompt wskazany w A.6 na stronie 114. Instruuje on model języka, aby rozważył czy konieczne jest wyszukanie informacji w internecie. Pytania ujawniające taką konieczność dotyczą aktualnych wydarzeń, wiadomości, sportu lub specjalistycznej wiedzy. prompt wymuszający część decyzyjną realizowanej strategii, wzboagacony jest o przykłady pytań z odpowiedzią negatywną - gdy pytanie uznane jest za niewymagające kontekstu oraz przeformułowań oryginalnych pytań na polecenie do wyszukania — gdy pytanie wymaga dodatkowych informacji. Przez klasę *SelfAware* obsługiwane jest odgałęzienie warunkowe na podstawie odpowiedzi modelu języka, gdyż od systemu potrzebującego pozyskania dodatkowych informacji oczekujemy innego zachowania niż od systemu potrafiącego odpowiedzieć na zadane pytanie bez kontekstu. Klasa *SelfAware* przyjmuje zatem dwa warianty obiektu *PIQARD* dostosowanych do konkretnego celu — odpowiedzi na pytanie bezpośrednio lub poszukiwania dodatkowych informacji. Za dalsze przetwarzanie, w zależności od

decyzji, odpowiada tylko jedna z instancji *PIQARD*. Podobnie jak w innych wariantach systemu za przechowywanie historii kolejnych etapów rozbudowanej strategii jest odpowiedzialna klasa *ChainTrace*.

Strategia *ReAct*

W strategii *ReAct* ważne są następujące elementy: klasa *Agent* reprezentująca model języka planujący sekwencję, zestaw dostępnych akcji, które agent może zaplanować, funkcje wykonujące zaplanowane akcje w postaci poleceń oraz środowisko dostarczające informacje.

Agent dziedziczy po klasie *PIQARD* i rozszerza jej wywołanie o realizację etapów strategii *ReAct*, takich jak wnioskowanie na podstawie obserwacji i planowanie sekwencji akcji potrzebnych do udzielenia odpowiedzi na zadane pytanie oraz generowanie poleceń do interakcji ze środowiskiem w celu pozyskania dodatkowych informacji.

Klasa *Action* jest interfejsem do wykonywania akcji, która jest zdefiniowana jako funkcja zwierająca kontekst na podstawie polecenia z zawartą kwerendą. Akcje rozpoznawalne są przez prefiks określany przy inicjalizacji obiektu. Wykonanie akcji odbywa się po jej wywołaniu podając ostatnią linię tekstu wygenerowaną przez model języka rozpoczynającej się od nazwy etapu przetwarzania "Action:". Jeśli zawiera ona słowo kluczowe *Search* oczekujemy, że wcześniej została zainicjowana akcja o identycznym prefiksie, która jest powiązana z klasą *Retrievera* umożliwiającą wyszukiwanie informacji i zwroćenie kontekstu dzięki metodzie *get_documents*.

Przebieg udzielania odpowiedzi na pytanie składa się z kilku powtarzających się etapów, dopóki nie zostanie wykryty koniec przetwarzania przez akcję *SequenceStopper* poszukującej polecenia o prefiksie *Finish*. Można wyróżnić trzy z nich:

- *thought* - łańcuch wnioskowania wygenerowany przez agenta
- *action* - akcja, polecenie z kwerendą wygenerowana przez agenta
- *observation* - obserwacja, wynik poprawnie wykonanej akcji

Łańcuch *ChainTrace* przechowuje informacje jako tekst zapisany w kolejnych węzłach o typie wskazującym na konkretny etap, w którym został wygenerowany lub wyszukany w zewnętrznym źródle danych (środowisku). Początkowo, w przetwarzaniu, model języka jest odpytywany bazowym promptem uzyskując odpowiedź pośrednią — docelowo jest to linia tekstu rozpoczynająca się od "Thought:", a kontynuacja tej linii zawiera zadanie do wykonania. Przykładowo dla pytania użytkownika: *What is the longest bridge in the world?*, odpowiedzią pośrednią może być: *Thought: I need to search the name of the longest bridge in the world.*

Po wygenerowaniu powyższego tekstu model powinien docelowo umieścić w kolejnej linii rozpoczynającej się od słowa kluczowego "Action:" wymaganą według niego akcję do wykonania. Iterując po zdefiniowanej wcześniej liście instancji klasy *Action* sprawdzane jest przy użyciu wyrażeń regularnych, czy istnieje obiekt z identycznym prefiksem, który może wykonać wygenerowane polecenie.

Następnie po wykonaniu żądanej przez model akcji, przekazywana jest zdobyta informacja po przez dodanie do kolejnego promptu linii rozpoczynającej się od słowa kluczowego "Observation:" z następującą po nich obserwacją. W ten sposób agent otrzymuje wymagane przez niego dane, które mają na celu wspomóc wygenerowanie odpowiedzi.

8.4.9 Obsługa konfiguracji yaml

YAML (skrót z ang. YAML Ain't Markup Language) jest językiem formalnym stosowanym do reprezentowania danych w ustrukturyzowany sposób, przy czym jednocześnie umożliwiając łatwy odczyt przez człowieka. Opierając się na minimalnej składni oraz wykorzystaniu wcięć linii do hierarchizacji danych, YAML bardzo dobrze nadaje się do tworzenia plików konfiguracyjnych i właśnie do tego go wykorzystujemy w naszym systemie. Pliki *config* służą do wyboru obiektów i parametrów używanych w trakcie działania aplikacji, a konfiguracja wybierana jest przy uruchomieniu aplikacji. W strukturze pliku wpierw podaje się nazwy klas wykorzystywanych w danej konfiguracji, a następnie w zagnieżdżeniu pod każdą z nich podaje się parametry ich inicjalizacji. Najważniejszą klasą do wskazania jest model języka, gdzie wybrać można jedną z wersji opisanych w 8.4.2. Aby wybrać metodę promptowania, wskazać trzeba odpowiedni *prompt_template*, a jeśli dodawane mają być też teksty uzupełniające kontekst wymagany jest również odpowiedni *information_retriever* wraz z nazwą wybranej bazy danych. Metod ReAct i Self-ask również wymagają podania dodatkowych parametrów takich jak funkcja wyszukująca informacje w internecie.

8.4.10 piqard_api

W ramach udostępnienia gotowej konfiguracji systemu stworzyliśmy plik, który umożliwia uruchomienie lokalnego serwera, wykorzystując wybrany adres IP oraz port. Korzystamy z gotowego rozwiązania zaproponowanego przez bibliotekę języka Python o nazwie *uvicorn*. Zawiera ona narzędzia i funkcje pozwalające na uruchomienie w prosty sposób serwera internetowego kompatybilnego z asynchronicznymi zapytaniami w standardzie ASGI (Asynchronous Server Gateway Interface). Aktualnie obsługuje ona protokół HTTP/1.1 oraz integracje z biblioteką WebSockets.

Skonfigurowany przez nas system odpowiadający na zadane pytanie z wyszukiwaniem kontekstu jest udostępniony dzięki API poprzez interakcje, z przygotowanym do odbierania danych, końcowym punktem dostępu. Oznacza to, iż aby uzyskać odpowiedź na pytanie zewnętrzny klient musi wysłać żądanie w protokole HTTP z metodą POST na adres URL serwera zakończony */opensystem*.

Do budowy systemu wykorzystaliśmy stworzoną przez nas bibliotekę, a dokładniej klasę *SelfAwareLoader*, która umożliwia zainicjowanie instancji klasy *SelfAware* z pliku konfiguracyjnego yaml. Dzięki temu mamy możliwość ewentualnej sprawnej zamiany elementów bazowych lub poszczególnych parametrów użytych rozwiązań. Poniżej przedstawiamy plik z zaproponowaną konfiguracją (8.15).

```

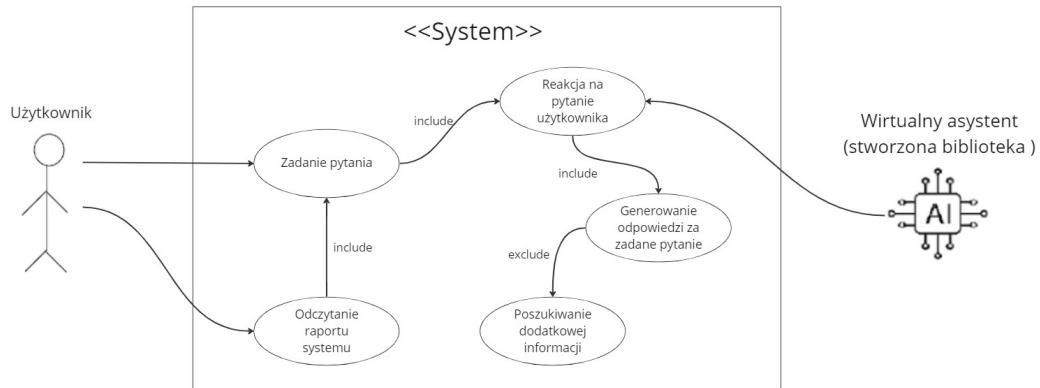
prompt_template: !PromptTemplate
  template: assets/prompting_templates/self_aware/self_aware_prompt.txt
language_model: !CohereAPI
  stop_token: "\n"
  temperature: 0
  top_k: 1
if_should_browse: !PIQARD
  prompt_template: !PromptTemplate
    template: assets/prompting_templates/like_chat_gpt/with_context.txt
language_model: !CohereAPI
  stop_token: "\n"
information_retriever: !WikiAPI
  k: 10
if_should_not_browse: !PIQARD
  prompt_template: !PromptTemplate
    template: assets/prompting_templates/like_chat_gpt/without_context.txt
language_model: !CohereAPI
  stop_token: "\n"

```

RYSUNEK 8.15: Plik yaml z proponowaną konfiguracją systemu opartego na klasie SelfAware.

8.4.11 piqard_app

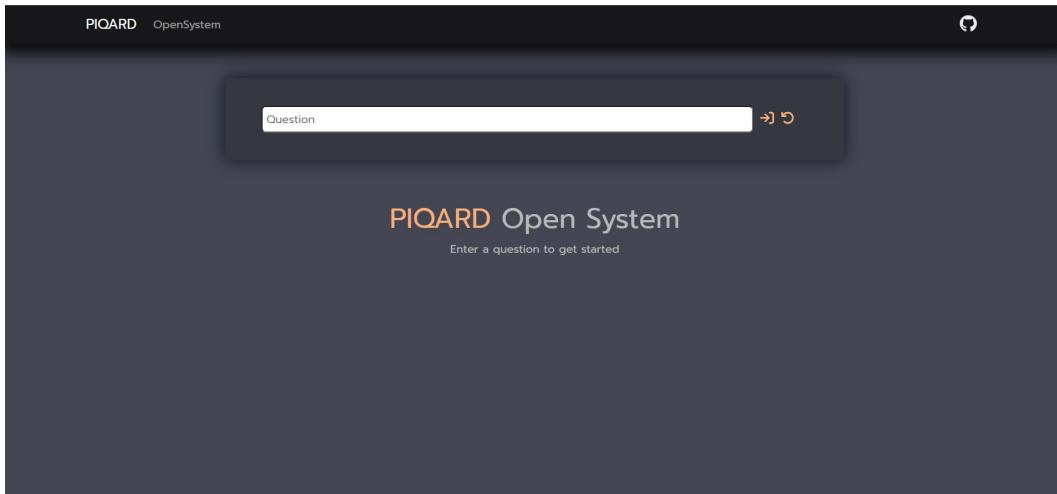
Piqard_app jest aplikacją internetową, która umożliwia użytkownikowi interakcje z proponowanym systemem poprzez stworzony interfejs graficzny. Została zaimplementowana w języku JavaScript w bibliotece React.js. Umożliwiło to nie tylko zaprojektowanie wyglądu i ułożenia elementów, ale także proste skonfigurowanie aplikacji internetowej, która jest uruchamiana na wybranym lokalnym porcie. Jej funkcjonalność bazuje na ustalonych wcześniej przypadkach użycia, które przedstawiamy w formie diagramu. (rys. 8.16)



RYSUNEK 8.16: Diagram przypadków użycia z rolami użytkownika oraz wirtualnego asystenta stworzonego z biblioteki.

W naszej pracy skupiliśmy się głównie na budowie biblioteki, która umożliwia złożenie systemu odpowiadającego na pytania z wyszukiwaniem kontekstu, dlatego w ramach interfejsu pozostały ograniczone funkcje. Ostatecznie użytkownik ma jedynie możliwość zadawania pytań oraz wyświetlenia odpowiedzi wraz z kolejnymi krokami, które doprowadziły do jej udzielenia. Umożliwia to widok, który w strukturze aplikacji nosi nazwę *OpenSystemView* (rys. 8.17 oraz rys. 8.18). Dodatkowo przygotowaliśmy stronę główną *HomeView* w celach reprezentacyjnych oraz zaznajomienia nowego użytkownika z proponowanym podejściem do rozwiązania problemu odpowiedzi na wielodziedzinowe pytania (rys. 8.19).

Widoki aplikacji były tworzone za pomocą funkcjonalnych komponentów, które od wersji Re-

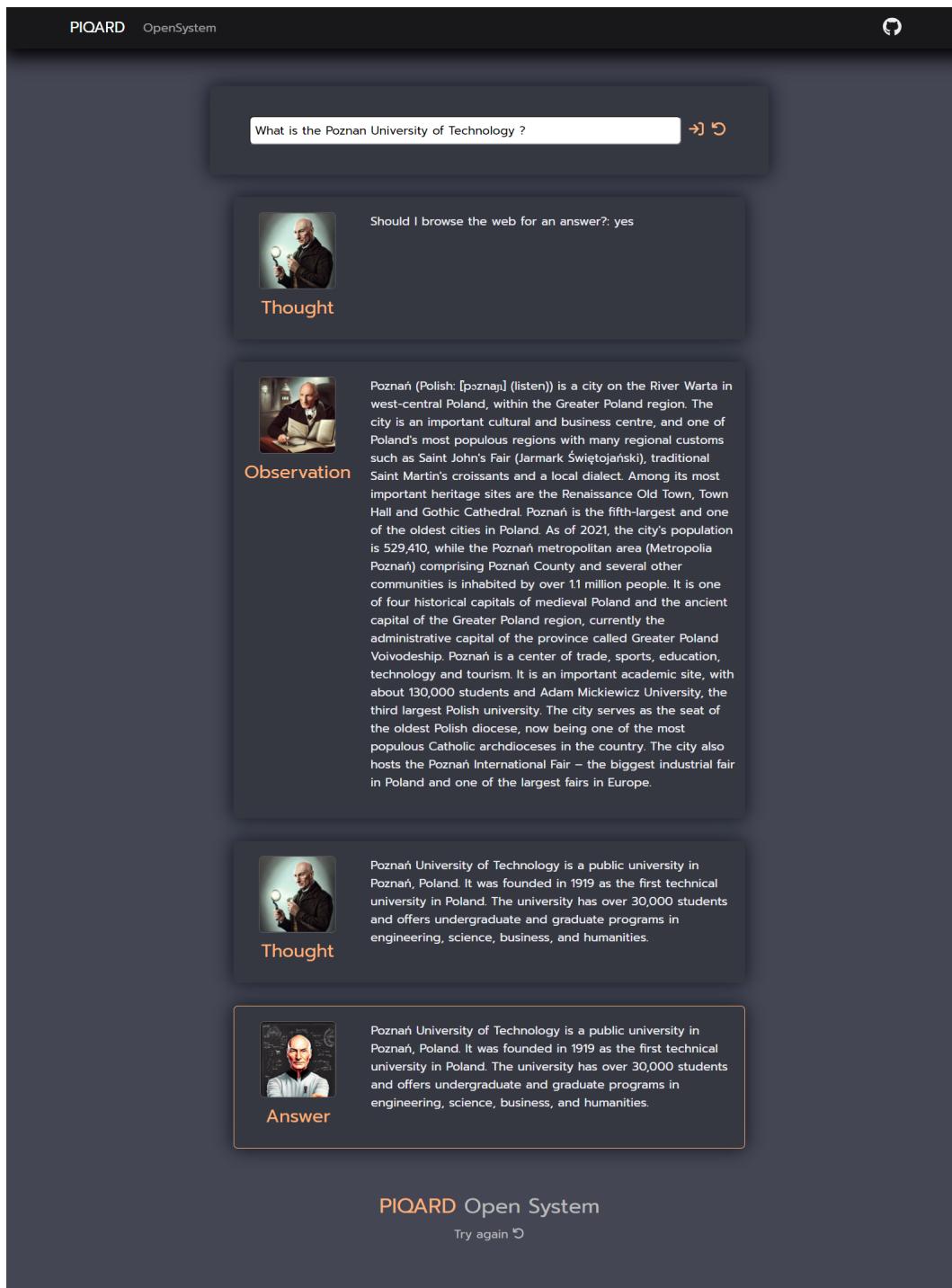


RYSUNEK 8.17: Widok ekranu, który pozwala użytkownikowi zadać pytanie.

act.js 16.8 stały się wiodącym sposobem implementacji elementów w składni jsx. Pozwala ona na umieszczanie kodu HTML w plikach JavaScript, co znacznie ułatwia i przyspiesza rozwój projektów. Warto także nadmienić, iż tworzone komponenty mogą przechowywać stany dzięki wprowadzeniu *React Hooks*, a sama aplikacja aktualizuje jedynie te elementy, których stan różni się od zapisanego w pamięci. Oznacza to, że nie występuje potrzeba ponownego ładowania strony za każdym razem, gdy użytkownik zmieni coś w jej statusie. W naszej pracy zastosowaliśmy także bibliotekę react-bootstrap, która ułatwia planowanie przestrzenne komponentów na stronie oraz zarządza nimi w podstawowy sposób w ramach responsywności, czyli odpowiedniemu dopasowaniu ułożenia zależnego od wielkości ekranu, na którym uruchomiono aplikację.

Analizując pliki projektu (rys. 8.9) można zauważyć, iż stworzyliśmy strukturę odzwierciedlającą budowane widoki w folderze *views*. Po za podziałem komponentów na komponenty ogólnego użycia oraz komponenty specyficzne dla danego widoku wydzieliliśmy także osobne katalogi na wykorzystywane zasoby, czcionki oraz wartości. Jeśli mowa o zasobach, są to ikony oraz obrazy umieszczone w aplikacji. Pierwsze z nich pochodzą głównie ze strony fontawesome.com, która udostępnia gotowe zestawy do wykorzystania we własnej pracy. Natomiast obrazy zostały wygenerowane przez nas dzięki technice *Stable Diffusion* [RBL⁺21] zastosowanej w pretrenowanym modelu o identycznej nazwie, który został udostępniony do darmowego użytku. Podając jako tekst wejściowy odpowiedni prompt uzyskaliśmy zadowalające nas wyniki, które wpasowały się w stylistykę projektu naszej aplikacji internetowej (rys. 8.20). Dodatkowo chcieliśmy zadbać o detale, dlatego wybraliśmy jedną z czcionek udostępnianych w bazach fonts.google.com, której nazwa to "*prompt*". Warto także wspomnieć, że folder *values* zawiera wartości wykorzystywane w plikach z rozszerzeniem css, które stylizują odpowiadające im pliki z rozszerzeniem jsx zawierające komponenty lub widoki.

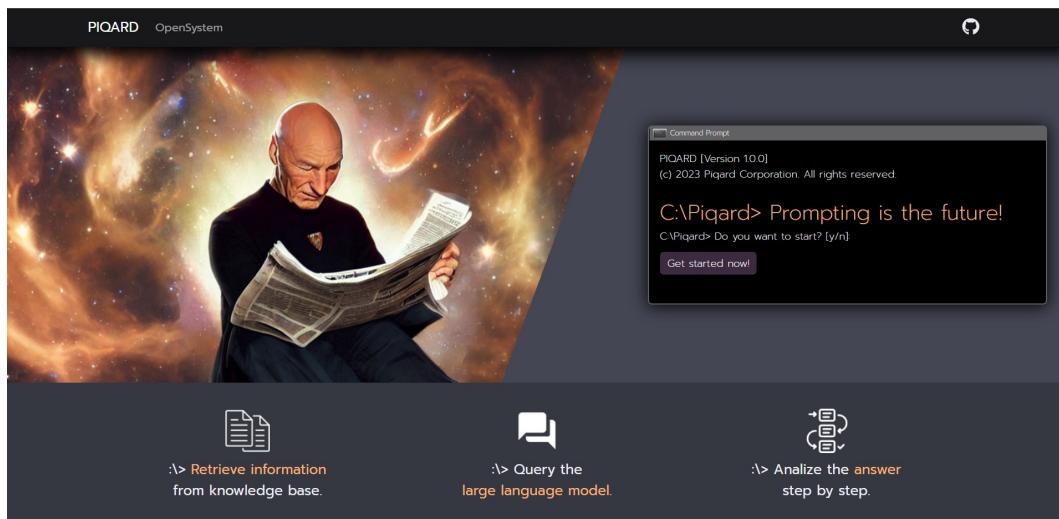
Integrację aplikacji internetowej z proponowanym systemem, umożliwiła biblioteka *axios*, która udostępnia funkcje odpowiadające za komunikację w protokole HTTP. Dzięki niej możliwe jest wysłanie żądania metodą POST z pytaniem użytkownika na adres IP oraz port, na którym znajduje się punkt dostępu z końcowym adresem URL */opensystem* uruchomionego wcześniej serwera *piqard_app*. Następnie otrzymywana jest odpowiedź systemu, która może zostać wyświetlona użytkownikowi.



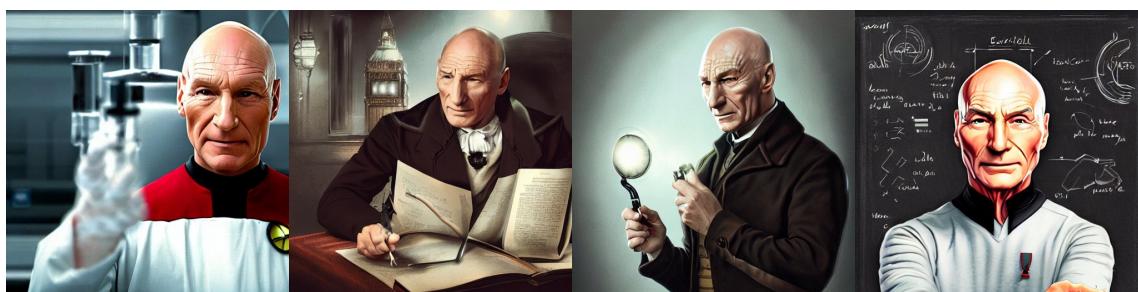
RYSUNEK 8.18: Widok ekranu, który pozwala użytkownikowi zadać pytanie wraz z otrzymaną odpowiedzią.

8.5 Obsługa błędów

Obsługę błędów w naszej pracy podzieliliśmy na dwa rodzaje. Pierwszy z nich dotyczy błędów możliwych do wystąpienia w trakcie korzystania ze stworzonej biblioteki. Wynikają one głównie z faktu, iż niektóre z jej komponentów jak np. CohereAPI lub GoogleCustomSearch wymagają komunikacji z zewnętrznymi API. Sprawia to, że nie mamy pełnej kontroli nad operacjami wykonywanymi po stronie dostawców, a co jest z tym związane, otrzymana odpowiedź serwera może



RYSUNEK 8.19: Widok ekranu głównego, z informacjami odnośnie systemu.



RYSUNEK 8.20: Przykłady wygenerowanych obrazów przez Stable Diffusion wykorzystanych w projekcie aplikacji internetowej dla możliwych do wykonania akcji systemu (od lewej: żądanie wykonania akcji, dokonanie obserwacji np. wyszukany kontekst, wygenerowanie częściowej odpowiedzi, wygenerowanie ostatecznej odpowiedzi).

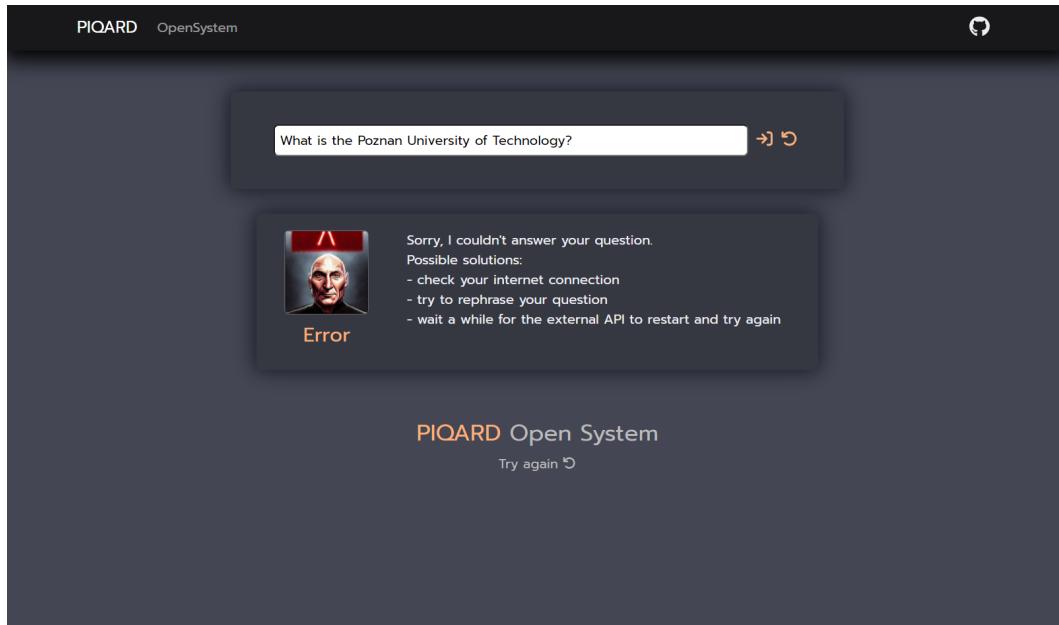
różnić się od oczekiwanej. Niechcianymi sytuacjami są, m.in. zwrócone komunikaty odnośnie limitów zapytań oraz błędy wynikające z niedostępności modeli, z racji awarii po stronie serwera. Większość z nich posiada osobne kody odpowiedzi protokołu HTTP, dzięki czemu możemy obsłużyć dany komunikat i zwrócić użytkownikowi wyjątek z jasno określona informacją. Przykładem jest odpowiedź serwera udostępniającego model Cohere xlarge, gdy generowany wynik zawiera tekst, który może naruszać zasady i politykę organizacji dotyczące, chociażby przemocy lub nierówności społecznych. Otrzymywana odpowiedź HTTP z serwera jest w takim wypadku sygnowana kodem 598, a po jej odebraniu podnosimy wyjątek o nazwie *LanguageModelAPIBlockedOutput* (rys. 8.21).

```
LanguageModelAPIBlockedOutput:
Cohere API -> Blocked output: this generation may be a potential violation of terms of service.
```

RYSUNEK 8.21: Wyjątek z komunikatem w konsoli jupyter notebook w przypadku możliwości naruszenia zasad i polityki organizacji w trakcie generowania odpowiedzi.

Drugi rodzaj błędów obsługiwany jest po stronie aplikacji internetowej. Z racji prostego interfejsu, w którym użytkownik może jedynie wpisać swoje zapytanie składające się z dowolnych znaków w pole tekstowe, istnieje tylko jeden komunikat. Wyświetla się on w momencie, gdy wystąpi błąd wynikający z użycia biblioteki i naruszy on działanie systemu w sposób krytyczny. Serwer zostanie w takim wypadku automatycznie zrestartowany, jednak aby użytkownik dowiedział się o

zaistniałej sytuacji przygotowaliśmy dla niego odpowiednią informację. Wyświetli się ona także, gdy serwer nie będzie aktywny, dzięki czemu użytkownik może go ręcznie uruchomić (rys. 8.22).



RYSUNEK 8.22: Komunikat z informacjami dotyczącymi braku udzielonej odpowiedzi przez system.

8.6 Testy

W ramach zapewnienia poprawności działania poszczególnych komponentów biblioteki oraz łatwego rozwoju i utrzymania tworzonych rozwiązań przygotowaliśmy testy jednostkowe. Polegają one na sprawdzeniu działania poszczególnych fragmentów kodu poprzez porównanie otrzymanych wyników z przygotowanymi przez programistę referencyjnymi wartościami. Dzięki takiemu podejęciu możliwe jest sprawne wykrycie ewentualnych błędów w implementacji i natychmiastowego naprawienia danych mankamentów.

Do stworzenia testów wykorzystaliśmy bibliotekę języka python o nazwie *PyTest*. Dzięki udostępnianym funkcjonalnością możliwe jest nie tylko uruchomienie przygotowanych zadań, ale także monitorowanie ich przebiegu i statystyk. Poniżej przedstawiamy wygenerowaną przez stosowaną bibliotekę tabelę 8.1 z wynikami pokrycia linii kodu dla stworzonych testów.

TABELA 8.1: Wyniki otrzymane za pomocą narzędzia PyTest pokrycia linii kodu dla postrzególnych plików. Kolumna o nazwie Name przedstawia nazwę pliku testowanego, kolumna o nazwie Stmts (Statements) reprezentuje ilość linii kodu do przetestowania, kolumna o nazwie Miss oznacza ilość niesprawdzonych linii kody, a o nazwie kolumna Cover reprezentuje procentowe pokrycie testami linii kodu

Name	Stmts	Miss	Cover
piqard\PIQARD.py	42	1	98%
piqard\PIQARD_loader.py	29	1	97%
piqard_init__.py	0	0	100%
piqard\extensions_init__.py	0	0	100%
piqard\extensions\dynamic_prompting.py	6	0	100%
piqard\information_retrievers\BM25_retriever.py	30	2	93%
piqard\information_retrievers_init__.py	6	0	100%
piqard\information_retrievers\annoy_retriever.py	26	0	100%
piqard\information_retrievers\faiss_retriever.py	23	2	91%
piqard\information_retrievers\google_custom_search.py	29	0	100%
piqard\information_retrievers\retriever.py	28	0	100%
piqard\information_retrievers\wiki_api.py	24	1	96%
piqard\language_models_init__.py	4	0	100%
piqard\language_models\bloom_176b_api.py	40	7	82%
piqard\language_models\cohere_api.py	32	5	84%
piqard\language_models\gpt_j6b_api.py	40	7	82%
piqard\language_models\language_model.py	9	0	100%
piqard_utils_init__.py	0	0	100%
piqard_utils\answer_postprocess.py	8	0	100%
piqard_utils\chain_trace.py	46	4	91%
piqard_utils\data_loaders_init__.py	2	0	100%
piqard_utils\data_loaders\database_loader_factory.py	6	0	100%
piqard_utils\data_loaders\hotpotqa_loader.py	17	0	100%
piqard_utils\data_loaders\openbookqa_loader.py	17	0	100%
piqard_utils\exceptions.py	35	0	100%
piqard_utils\io.py	21	0	100%
piqard_utils\jinja_loader.py	10	0	100%
piqard_utils\prompt_template.py	16	0	100%
piqard_utils\yaml_constructor.py	15	2	87%
tests\piqard\extensions\test_dynamic_prompting.py	6	0	100%
tests\piqard\information_retrievers\test_annoy_retriever.py	27	0	100%
tests\piqard\information_retrievers\test_bm25_retriever.py	81	0	100%
tests\piqard\information_retrievers\test_faiss_retriever.py	55	0	100%
tests\piqard\information_retrievers\test_google_custom_search.py	16	0	100%
tests\piqard\information_retrievers\test_retriever.py	38	0	100%
tests\piqard\information_retrievers\test_wikiapi.py	24	0	100%
tests\piqard\language_models\test_bloom176b_api.py	21	0	100%
tests\piqard\language_models\test_cohere_api.py	15	0	100%
tests\piqard\language_models\test_gpt_j6b.py	19	0	100%
tests\piqard\language_models\test_language_model.py	8	0	100%
tests\piqard\test_piqard.py	70	0	100%
tests\piqard\test_piqard_loader.py	15	0	100%
tests\piqard_utils\data_loaders\test_database_loader_factory.py	16	2	88%
tests\piqard_utils\data_loaders\test_hotpotqa_loader.py	32	0	100%
tests\piqard_utils\data_loaders\test_openbookqa_loader.py	28	0	100%
tests\piqard_utils\test_answer_postprocess.py	14	0	100%
tests\piqard_utils\test_chain_trace.py	41	0	100%
tests\piqard_utils\test_exceptions.py	27	0	100%
tests\piqard_utils\test_io.py	42	0	100%
tests\piqard_utils\test_jinja_loader.py	14	0	100%
tests\piqard_utils\test_prompt_template.py	21	0	100%
tests\piqard_utils\test_yaml_constructor.py	0	0	100%
TOTAL	1191	34	97%

Rozdział 9

Eksperymenty

W ramach naszej pracy przeprowadziliśmy eksperymenty, które miały na celu sprawdzenie wpływu określonych strategii promptingu na jakość generowanych odpowiedzi. W kolejnych podrozdziałach przedstawimy przebieg oraz wyniki poszczególnych testów:

- **Ocena wpływu liczby dokumentów podanych jako kontekst dla pytania testowego**
- eksperyment ma na celu określenie, jak zmienia się wartość wybranych miar oceny wraz ze zwiększeniem liczby wyszukanych dokumentów. Dodatkowo porównane zostały metody leksykalnego oraz semantycznego wyszukiwania informacji, w celu próby wyłonienia najlepszego z algorytmów BM25, FAISS oraz ANNOY.
- **Ocena wpływu liczby podanych przykładów rozwiązania zadania w promptcie** - eksperyment ma na celu określenie, jak zmienia się wartość wybranych miar oceny wraz ze zwiększeniem liczby podanych przykładów udzielenia odpowiedzi na pytanie.
- **Ocena działania strategii promptingu nazwanej łańcuchem wnioskowania** - eksperyment ma na celu sprawdzenie jakości generowanych odpowiedzi z wykorzystaniem promptu opartego na strategii łańcucha wnioskowania, który ma za zadanie wymusić na modelu, aby podał kolejne kroki prowadzące do ostatecznego wyniku.
- **Ocena działania strategii nazwanej dynamicznym promptingiem** - eksperyment ma na celu sprawdzenie jakości generowanych odpowiedzi z wykorzystaniem promptu, w którym dla każdego pytania testowego wyszukiwane są przykłady ze zbioru treningowego.
- **Ocena wpływu kolejności występowania przykładów w promptcie** - eksperyment ma na celu przetestowanie, czy zmiana kolejności przykładów w promptcie wpływa na wynik generowanej odpowiedzi

Do przeprowadzenia eksperymentów wybraliśmy dwa benchmarki OpenBookQA oraz HotpotQA. Pierwszy z nich zwiera pytania wielokrotnego wyboru i dotyczy głównie faktów naukowych, natomiast drugi składa się z pytań otwartych, których tematyka jest wielodziedzinowa. Dla obu z nich autorzy udostępnili zbiory dokumentów, które posłużyły nam do wyszukiwania kontekstu dla poszczególnych pytań testowych. Ich szerszą analizę przeprowadziliśmy w rozdziale 7, jednak chcielibyśmy wspomnieć i doprecyzować, iż do wykonania eksperymentów opisanych w dalszej części wybraliśmy zbiór testowy pięciuset pytań OpenBookQA określany przez twórców jako *test questions* oraz pięćset pierwszych pytań ze zbioru walidacyjnego HotpotQA określany przez twórców jako *dev questions*. Druga decyzja została podkutowana ograniczeniami zasobów sprzętowych

i obliczeniowych, które sprawiły, iż musielibyśmy zredukować liczbę pytań. Jednocześnie, aby umożliwić obliczenie lokalnie wybranych miar oceny, pozostałyśmy przy zbiorze walidacyjnym, który w przeciwnieństwie do zbioru testowego zawiera także poprawne odpowiedzi.

Wspominając o miarach oceny, warto zauważyć, że wybrane przez nas różnią się od oryginalnie zdefiniowanych dla poszczególnych benchmarków. Twórcy OpenBookQA stosują w swojej pracy miarę zwana trafnością, którą w naszym przypadku zamieniliśmy na dokładne dopasowanie. Zastosowanie dokładnego dopasowania w przypadku generowania odpowiedzi ma identyczne wartości jak obliczana na tych samych wynikach trafność. Dodatkowo rozszerzyliśmy ją o pokrycie dokładnego dopasowania i miarę F1. Jeśli mowa o benchmarku HotpotQA, twórcy zaproponowali dokładne dopasowanie oraz miarę F1, które dodatkowo rozszerzyliśmy o pokrycie dokładnego dopasowania oraz BLEU w wersji dla pojedynczych, podwójnych oraz potrójnych n-gramów.

Przeprowadzenie eksperymentów dla poszczególnych strategii promptingu umożliwiła stworzona przez nas biblioteka, w której zaimplementowaliśmy klasy udostępniające wykorzystanie modeli językowych BLOOM 176B (dalej zwanym BLOOM) oraz Cohere xlarge. Wybraliśmy je, m.in. z powodu dostępnych API pozwalających na ich odpytanie bez potrzeby posiadania lokalnie ogromnych zasobów obliczeniowych. Dodatkowo zauważamy, iż aktualnie w literaturze naukowej nie przeprowadzono stosunkowo dużo testów z ich udziałem, dzięki czemu możemy wzbogacić dostępne o nich informacje. Zaproponowana przez nas biblioteka zapewnia także metody wyszukiwania informacji BM25, FAISS i ANNOY, które znalazły swoje zastosowanie w niektórych z testów. W celu ich użycia zbudowaliśmy indeksy dla benchmarku OpenBookQA oraz HotpotQA wykorzystując model dostrajany w technice sentence-transformer nazwany *multi-qa-MiniLM-L6-cos-v1*, dla którego wymiar danych wynosi 384, a miara odległości została ustalona na kosinusową. Dodatkowo dla indeksu stworzonego dla FAISS o typie *IndexIVFPQ* wybraliśmy podział wektorów na 8 segmentów, a dla ANNOY wartość 100 tworzonych drzew.

Poniżej przedstawiamy szczegółowe opisy przebiegu eksperymentów wraz z wynikami oraz sformułowanymi na ich podstawie wnioskami.

9.1 Ocena wpływu liczby dokumentów podanych jako kontekst dla pytania testowego

Celem eksperymentu jest przetestowanie wpływu liczby wyszukanych dokumentów podanych jako kontekst dla pytania na jakość wygenerowanych odpowiedzi. Dodatkowo porównane zostaną metody leksykalnego i semantycznego wyszukiwania informacji, jako próba wyłonienia najlepszego ze sposobów BM25, FAISS oraz ANNOY. Do jego przeprowadzenia przygotowaliśmy specjalny prompt składający się z prostej instrukcji “Answer the question based on the facts.” oraz z pięciu przykładów odpowiedzi na pytanie wraz z kontekstem (patrz tab. A.10 w załączniku). Domyślnie kontekst stanowią manualnie dopasowane trzy dokumenty dla każdego pytania. Zmiennym parametrem w tym eksperymencie jest ich liczba znajdująca się w przedziale $<0,3>$, do której redukowaliśmy podane dla każdego przykładu dokumenty, rozpoczynając od ich końca.

9.1.1 Opis konfiguracji systemu

Eksperyment przeprowadziliśmy dla benchmarku OpenBookQA w sześciu różnych konfiguracjach systemu. Przetestowaliśmy wybrane modele języka, czyli BLOOM oraz Cohere xlarge, dla których ustaliliśmy parametr *temperature* na wartość 1e-6 dla pierwszego modelu i 0 dla kolejnego oraz *top_k* na wartość 1, co wpływa na większy determinizm generowanych odpowiedzi. Sprawdzi-

liśmy ich działanie w połączeniu z każdą z trzech metod wyszukiwania informacji tj. BM25, FAISS oraz ANNOY.

9.1.2 Wyniki

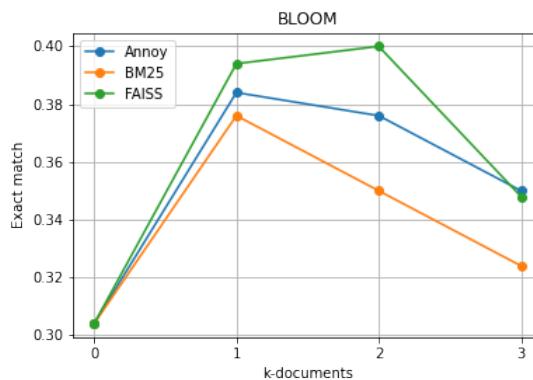
Wyniki dla poszczególnych miar oceny przedstawiamy w tabelach 9.1, 9.2, 9.3 dotyczących BLOOM oraz tabelach 9.4, 9.5, 9.6 dotyczących Cohere xlarge. Dodatkowo przygotowaliśmy ich wizualizacje w formie wykresów.

BLOOM

Najlepszy wynik dokładnego dopasowania osiągnęliśmy dla konfiguracji z modelem języka BLOOM w połączeniu z metodą wyszukiwania FAISS i wynosi on 0.4 przy wyszukaniu dwóch dokumentów podanych następnie jako kontekst dla danego pytania testowego (tab. 9.1). Jak można zauważyć na wykresie (rys. 9.1), dla wszystkich trzech metod wyszukiwania informacji, wartości dokładnego dopasowania początkowo rosną, lecz wykorzystując metodę BM25 oraz ANNOY zaczynają maleć już przy wybraniu dwóch dokumentów, a dla metody FAISS przy wybraniu trzech dokumentów.

TABELA 9.1: Wartości dokładnego dopasowania w zależności od liczby k wyszukanych dokumentów dla modelu języka BLOOM oraz metod wyszukiwania informacji BM25, FAISS i ANNOY.

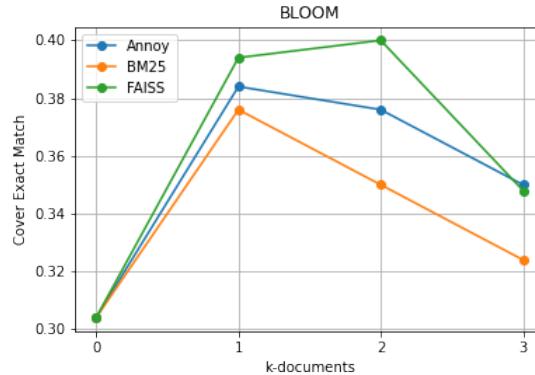
	k = 0	k = 1	k = 2	k = 3
BM25	0.304	0.376	0.35	0.324
FAISS	0.304	0.394	0.4	0.348
ANNOY	0.304	0.384	0.376	0.35



RYSUNEK 9.1: Wykres zależności dokładnego dopasowania (ang. exact match) od liczby wyszukanych dokumentów (k-documents) dla modelu języka BLOOM.

TABELA 9.2: Wartości pokrycia dokładnego dopasowania w zależności od liczby k wyszukanych dokumentów dla modelu języka BLOOM oraz metod wyszukiwania informacji BM25, FAISS i ANNOY.

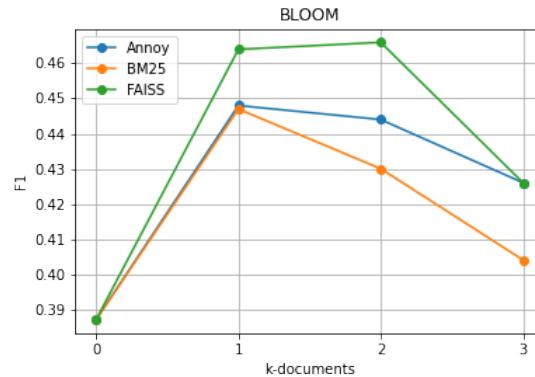
	k = 0	k = 1	k = 2	k = 3
BM25	0.304	0.376	0.35	0.324
FAISS	0.304	0.394	0.4	0.348
ANNOY	0.304	0.384	0.376	0.35



RYSUNEK 9.2: Wykres zależności pokrycia dokładnego dopasowania (ang. cover exact match) od liczby wyszukanych dokumentów (k-documents) dla modelu języka BLOOM.

TABELA 9.3: Wartości miary F1 w zależności od liczby k wyszukanych dokumentów dla modelu języka BLOOM oraz metod wyszukiwania informacji BM25, FAISS i ANNOY.

	k = 0	k = 1	k = 2	k = 3
BM25	0.387	0.447	0.43	0.404
FAISS	0.387	0.464	0.466	0.426
ANNOY	0.387	0.448	0.444	0.426



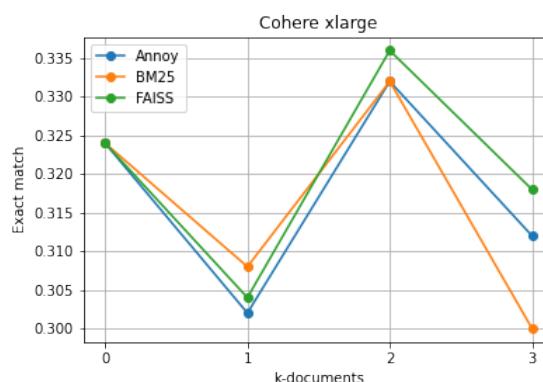
RYSUNEK 9.3: Wykres zależności miary F1 (F1) od liczby wyszukanych dokumentów (k-documents) dla modelu języka BLOOM.

Cohere xlarge

W przypadku konfiguracji z modelem języka Cohere xlarge najlepszy wynik dokładnego dopasowania wynosi **0.336**. Podobnie jak dla poprzedniego modelu, osiągnęliśmy go w konfiguracji z metodą wyszukiwania informacji FAISS dla dwóch wyszukanych dokumentów podanych jako kontekst danego pytania testowego. Widoczny jest również podobny trend do konfiguracji z modelem języka BLOOM, w którym wynik z trzema dokumentami jest znacznie mniejszy niż z dwoma. Zaskakujący okazuje się jednak fakt, iż dla jednego wyszukanego dokumentu miara dokładnego dopasowania jest mniejsza niż w przypadku braku podanego kontekstu (zero wyszukanych dokumentów). Niestety organizacja Cohere nie podaje szczegółów dotyczących wykorzystanego zbioru treningowego, jednak istnieje prawdopodobieństwo, iż zawarte w nim dokumenty sprawiły, że model zapamiętał w swoich wagach informacje potrzebne do odpowiedzi na pytania z benchmarku OpenBookQA. W takim przypadku model Cohere xlarge powinien móc odpowiedzieć poprawnie na niektóre pytania testowe z tego zbioru nawet bez podania dodatkowego kontekstu. Zakładając powyższe oraz fakt, że pierwszy wyszukany dokument nie zawsze jest idealnie dopasowany do pytania testowego, czego powodem jest niedoskonałość użytych metod wyszukiwania oraz dokonanej wektoryzacji dokumentów, istnieje możliwość, że wskazany kontekst zawiera niepotrzebne w danym przypadku informacje. Sprawia to, iż model zostaje wprowadzony w błąd i udziela złej odpowiedzi. Warto dodać, że wraz ze wzrostem liczby wyszukanych dokumentów, wzrasta szansa, aby w kontekście pojawił się dokument najbardziej informatywny dla danego pytania.

TABELA 9.4: Wartości dokładnego dopasowania w zależności od liczby k wyszukanych dokumentów dla modelu języka Cohere xlarge oraz metod wyszukiwania informacji BM25, FAISS i ANNOY.

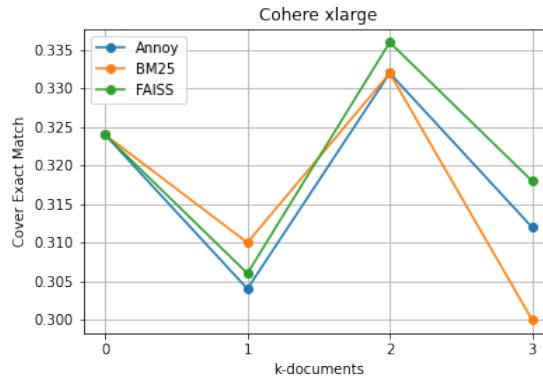
	k = 0	k = 1	k = 2	k = 3
BM25	0.324	0.308	0.332	0.3
FAISS	0.324	0.304	0.336	0.318
ANNOY	0.324	0.302	0.332	0.312



RYSUNEK 9.4: Wykres zależności dokładnego dopasowania (ang. exact match) od liczby wyszukanych dokumentów (k-documents) dla modelu języka Cohere xlarge.

TABELA 9.5: Wartości pokrycia dokładnego dopasowania w zależności od liczby k wyszukanych dokumentów dla modelu języka Cohere xlarge oraz metod wyszukiwania informacji BM25, FAISS i ANNOY.

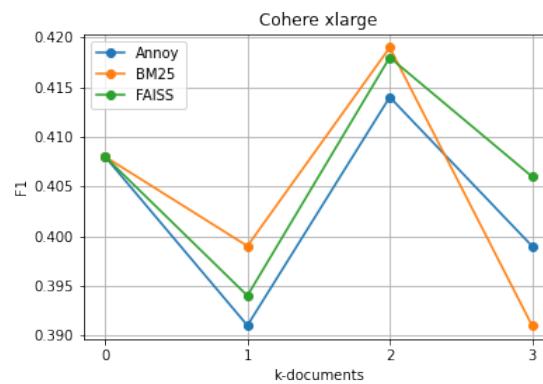
	k = 0	k = 1	k = 2	k = 3
BM25	0.324	0.31	0.332	0.3
FAISS	0.324	0.306	0.336	0.318
ANNOY	0.324	0.304	0.332	0.312



RYSUNEK 9.5: Wykres zależności pokrycia dokładnego dopasowania (ang. cover exact match) od liczby wyszukanych dokumentów (k-documents) dla modelu języka Cohere xlarge.

TABELA 9.6: Wartości miary F1 w zależności od liczby k wyszukanych dokumentów dla modelu języka Cohere xlarge oraz metod wyszukiwania informacji BM25, FAISS i ANNOY.

	k = 0	k = 1	k = 2	k = 3
BM25	0.408	0.399	0.419	0.391
FAISS	0.408	0.394	0.418	0.406
ANNOY	0.408	0.391	0.414	0.399



RYSUNEK 9.6: Wykres zależności miary F1 (F1) od liczby wyszukanych dokumentów (k-documents) dla modelu języka Cohere xlarge.

9.1.3 Wnioski szerszej analizy wyników

Wyniki eksperymentu wskazują, że niezależnie od konfiguracji, dodając do promptu kontekst w formie jednego, dwóch lub trzech dokumentów można poprawić wyniki generowanych odpowiedzi w porównaniu z brakiem podanego kontekstu dla pytania. Wyjątkiem jest wyszukanie jednego i trzech dokumentów w przypadku konfiguracji z modelem języka Cohere xlarge, o czym wspomnialiśmy

powyżej. Oznacza to, że wyszukany według podobieństwa do pytania użytkownika kontekst, który następnie przekazany jest dodatkowo jako dane wejściowe dla modelu języka, zazwyczaj wpływa pozytywnie na jego działanie.

Porównując osiągnięte wyniki dla dokładnego dopasowania w konfiguracjach z modelem języka BLOOM (tab. 9.1) oraz Cohere xlarge (tab. 9.4) można zauważać znaczną przewagę pierwszego z nich. Niezależnie od liczby wyszukanych dokumentów podanych jako kontekst model językowy BLOOM generował lepsze odpowiedzi. Dodatkowa różnica między najlepszym osiągniętym wynikiem przez BLOOM a najlepszym osiągniętym wynikiem przez Cohere xlarge wynosi 0.064 na korzyść pierwszego z nich. Może to być spowodowane wspomnianymi wcześniej obserwacjami dotyczącymi prawa skalowalności dużych modeli [KMH⁺20], gdyż BLOOM posiada ponad 3 razy więcej parametrów, czyli wag połączeń, niż Cohere xlarge. Warto jednak zauważyć, że dla promptu, w którym nie dodano kontekstu (wyszukano zero dokumentów) model języka Cohere xlarge osiągnął lepszy wynik niż BLOOM. Może to być spowodowane specyfiką uczenia i różnicą zawartości zbiorów treningowych. Jeśli model Cohere xlarge w trakcie treningu miał dostęp do danych, które zawierały przydatne w kontekście benchmarku OpenBookQA informacje oraz zapamiętał je w swoich parametrach (wagach połączeń), istnieje szansa, aby odpowiadał lepiej bez dodatkowego kontekstu niż jego konkurent BLOOM trenowany głównie na zbiorze wielojęzycznych dokumentów z popularnych stron internetowych oraz plikach kodu z językami programowania.

Eksperyment wykazał także, iż w przypadku wybranych benchmarków oraz modeli języka najlepsze wyniki osiągaliśmy dla konfiguracji z metodami wyszukiwania semantycznego. Najwyższą miarę dokładnego dopasowania zazwyczaj osiągnął FAISS, w związku z tym testy w dalszej części rozdziału zostaną przeprowadzone jedynie z wykorzystaniem tego wariantu.

9.2 Ocena wpływu liczby podanych przykładów rozwiązania zadania w promptcie

Celem eksperymentu jest sprawdzenie wpływu liczby podanych przykładów rozwiązania zadania w promptcie na jakość generowanej odpowiedzi. Do jego przeprowadzenia stworzyliśmy dwa podstawowe wzorce. Wzorzec dla benchmarku OpenBookQA składa się z prostej instrukcji “Answer the question based on the facts.” oraz z pięciu przykładów odpowiedzi na pytanie wraz z podanym jednym dokumentem jako kontekst (patrz tab. A.9 w załączniku). Natomiast wzorzec dla benchmarku HotpotQA składa się z instrukcji “Given the following question and contexts provide an answer:” oraz z pięciu przykładów odpowiedzi na pytanie wraz z dwoma podanymi dokumentami jako kontekst (patrz tab. A.14 w załączniku). Zmiennym parametrem w tym eksperymencie jest liczba podanych przykładów z przedziału $<0,5>$, dlatego w zależności od niej usuwaliśmy ich odpowiednią liczbę, rozpoczynając od końca promptu.

9.2.1 Opis konfiguracji systemu

Eksperyment przeprowadziliśmy dla benchmarku OpenBookQA oraz HotpotQA w dwóch różnych konfiguracjach systemu. Przetestowaliśmy wybrane modele języka, czyli BLOOM oraz Cohere xlarge, dla których ustaliliśmy parametr *temperature* na wartość $1e-6$ dla pierwszego modelu i 0 dla kolejnego oraz *top_k* na wartość 1 , co wpływa na większy determinizm generowanych odpowiedzi. Sprawdziliśmy ich działanie w połączeniu z metodą wyszukiwania informacji FAISS.

9.2.2 Wyniki

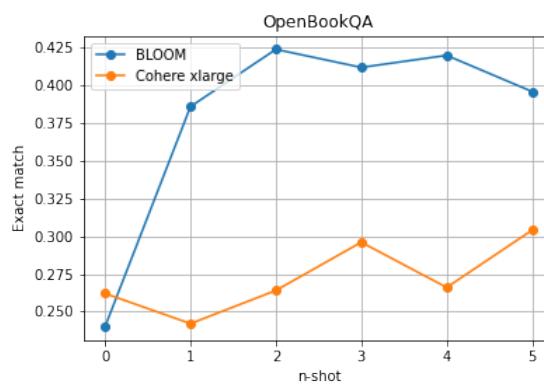
Wyniki dla poszczególnych miar oceny przedstawiamy poniżej. Tabele 9.7, 9.8, 9.9 dotyczą testów przeprowadzonych na benchmarku OpenBookQA dla modeli BLOOM i Cohere xlarge. Natomiast tabele 9.10, 9.11, 9.12, 9.13, 9.14, 9.15, dotyczą testów przeprowadzonych na benchmarku HotpotQA wykorzystując te same modele. Dodatkowo przygotowaliśmy wizualizacje wszystkich wyników w formie wykresów.

OpenBookQA

Najlepszy wynik dokładnego dopasowania osiągnęliśmy dla konfiguracji z modelem języka BLOOM i wynosi on *0.424* przy podaniu dwóch przykładów w promptcie (tab. 9.7). Jak wskazują charakterystyki wykresów (rys. 9.7), model ten osiąga generalnie lepsze wyniki od Cohere xlarge niezależnie od ich wybranej liczby. Wyjątkiem jest podejście 0-shot (niepodanie żadnego przykładu), dla którego podobnie jak w poprzednim eksperymencie drugi z modeli radzi sobie lepiej. Brak informacji ze strony twórców Cohere o szczegółach zbioru treningowego i samego uczenia ponownie sprawia trudność w określeniu przyczyny takiego zachowania. Warto także zauważyć, że w przypadku modelu języka BLOOM strategia promptingu z podaniem kilku przykładów odpowiedzi poprawia znacznie jego wyniki. Dokładne dopasowanie zwiększa się aż o *0.184* między promptem z brakiem podanych przykładów, a jego wersją z dwoma przykładami i następnie oscyluje przy wartości *0.412*. W przeciwieństwie do poprzednika różnica dla modelu Cohere xlarge jest znacznie mniejsza, choć nadal można zaobserwować poprawę jego wyników. Wartość dokładnego dopasowania w tym przypadku zmieniła się z *0.262* (brak podanych przykładów), jedynie do *0.306* (pięć podanych przykładów).

TABELA 9.7: Wartości dokładnego dopasowania w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

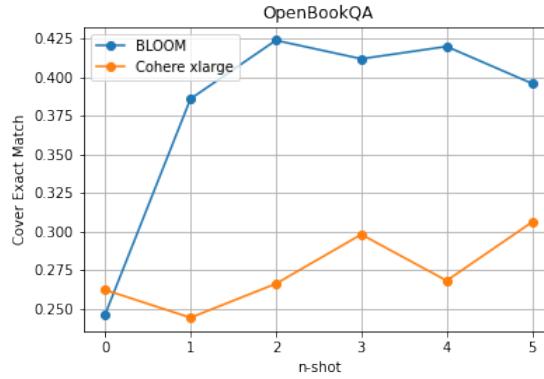
	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.24	0.386	0.424	0.412	0.42	0.396
Cohere xlarge	0.262	0.242	0.264	0.296	0.266	0.304



RYSUNEK 9.7: Wykres zależności dokładnego dopasowania (ang. exact match) od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.8: Wartości pokrycia dokładnego dopasowania w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

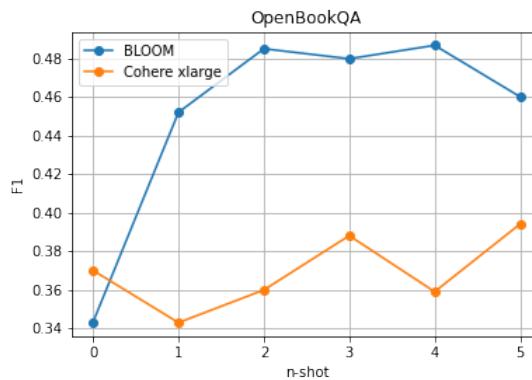
	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.246	0.386	0.424	0.412	0.42	0.396
Cohere xlarge	0.262	0.244	0.266	0.298	0.268	0.306



RYSUNEK 9.8: Wykres zależności pokrycia dokładnego dopasowania (ang. cover exact match) od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.9: Wartości miary F1 w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.343	0.452	0.485	0.48	0.487	0.46
Cohere xlarge	0.37	0.343	0.36	0.388	0.359	0.394



RYSUNEK 9.9: Wykres zależności miary F1 (F1 score) od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

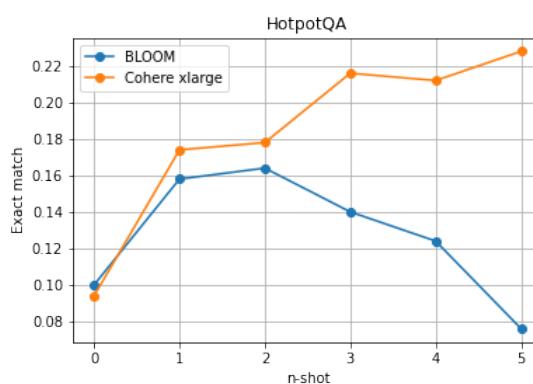
HotpotQA

Najlepszy wynik dokładnego dopasowania osiągnęliśmy dla konfiguracji z modelem języka Cohere xlarge i wynosi on **0.228** przy podaniu pięciu przykładów w promptcie (tab. 9.10). Zastosowana strategia promptingu poprawia jego wyniki niezależnie od ich wybranej liczby, a trend zaobserwowany na wykresie jest zdecydowanie wzrostowy (rys. 9.10). Wpływ liczby podanych przykładów na wyniki dla modelu BLOOM także początkowo jest pozytywny. Niestety po osiągnięciu maksymalnej wartości dokładnego dopasowania **0.164** dla dwóch podanych przykładów, następuje zmiana trenu na przeciwny i wartości maleją. Ostatecznie osiągany jest wynik nawet poniżej początkowej wartości dla promptu z pięcioma podanymi przykładami.

Warto zauważyć, że w obu przypadkach miara oceny pokrycia dokładnego dopasowania jest początkowo wysoka i wynosi około **0.2** (tab. 9.11), a wartość dokładnego dopasowania to około **0.1** (tab. 9.10). Świadczy to o tym, iż wygenerowana przez modele odpowiedź często zawierała wszystkie słowa występujące w odpowiedzi referencyjnej, lecz znajdowały się one w sąsiedztwie dodatkowych, niepotrzebnych wyrazów. Następnie wraz ze wzrostem liczby podanych przykładów można zaobserwować utrzymanie podobnego wyniku pokrycia dokładnego dopasowania dla modelu Cohere xlarge, z jednoczesną poprawą dokładnego dopasowania prawie do identycznego poziomu. Ukazuje to, że Cohere xlarge zaadaptował się do formy oczekiwanej odpowiedzi na podstawie podanych przykładów i dostosował do niej generowanie wyników. Dla modelu BLOOM także można zaobserwować początkowe utrzymanie wyniku pokrycia dokładnego dopasowania na podobnym poziomie, w trakcie jednoczesnego wzrostu wartości dokładnego dopasowania. Oznacza to, że drugi model również uczył się na podstawie przykładów i dostosowywał formę odpowiedzi do obowiązującej w benchmarku. Sytuacja ta ulega jednak zmianie po osiągnięciu maksymalnej wartości dla wersji promptu z podanymi dwoma przykładami. Od tego punktu zarówno wyniki pokrycia dokładnego dopasowania, jak i wyniki dokładnego dopasowania z każdym kolejnym przykładem maleją o podobną wartość. W naszej opinii można przypuszczać, że ich dodawanie do promptu sprawiło, iż BLOOM zaczął generować odpowiedzi zawierające całkowicie inną treść.

TABELA 9.10: Wartości dokładnego dopasowania w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

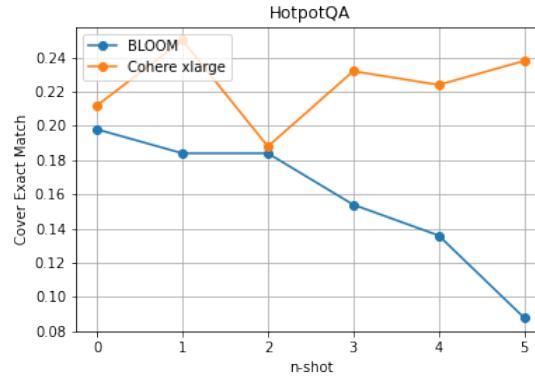
	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.1	0.158	0.164	0.14	0.124	0.076
Cohere xlarge	0.094	0.174	0.178	0.216	0.212	0.228



RYSUNEK 9.10: Wykres zależności dokładnego dopasowania (ang. exact match) od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.11: Wartości pokrycia dokładnego dopasowania w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

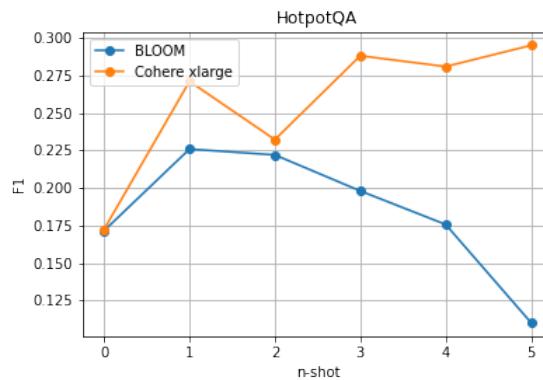
	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.198	0.184	0.184	0.154	0.136	0.088
Cohere xlarge	0.212	0.25	0.188	0.232	0.224	0.238



RYSUNEK 9.11: Wykres zależności pokrycia dokładnego dopasowania (ang. cover exact match) od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.12: Wartości miary F1 w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

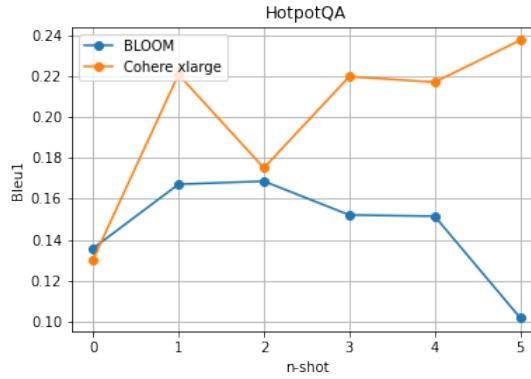
	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.171	0.226	0.222	0.198	0.176	0.11
Cohere xlarge	0.172	0.271	0.232	0.288	0.281	0.295



RYSUNEK 9.12: Wykres zależności miary F1 (F1 score) od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.13: Wartości BLEU-1 w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.136	0.167	0.169	0.152	0.152	0.102
Cohere xlarge	0.13	0.221	0.175	0.22	0.217	0.237



RYSUNEK 9.13: Wykres zależności BLEU-1 od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.14: Wartości BLEU-2 w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

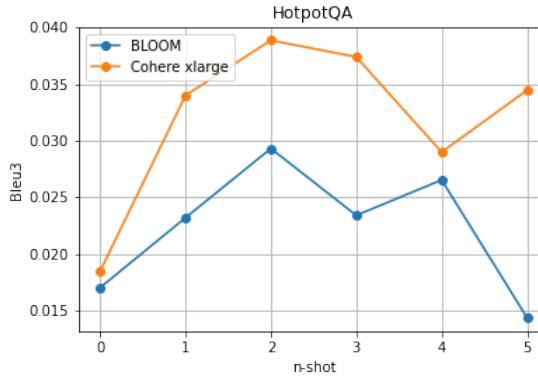
	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.081	0.097	0.101	0.092	0.095	0.061
Cohere xlarge	0.091	0.138	0.121	0.146	0.133	0.149



RYSUNEK 9.14: Wykres zależności BLEU-2 od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.15: Wartości BLEU-3 w zależności od liczby n podanych przykładów dla modelu języka BLOOM oraz Cohere xlarge.

	n = 0	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.017	0.023	0.029	0.023	0.027	0.014
Cohere xlarge	0.018	0.034	0.039	0.037	0.029	0.034



RYSUNEK 9.15: Wykres zależności BLEU-3 od liczby podanych przykładów (n-shot) dla modelu języka BLOOM oraz Cohere xlarge.

9.2.3 Wnioski szerszej analizy wyników

Wyniki eksperymentu wskazują, że niezależnie od konfiguracji, dodanie do promptu liczby przykładów z przedziału $<1, 5>$ na ogół poprawia osiągane wyniki generowanych odpowiedzi w porównaniu z wersją promptu, która ich nie zawiera. Trend ten nie jest jednak zawsze wzrostowy i jak ukazuje przykład modelu BLOOM, mimo początkowych wzrostów wyniku dokładnego dopasowania na benchmarku HotpotQA, jego wartości zaczęły stopniowo maleć po przekroczeniu dwóch przykładów.

Zaskakująca okazuje się obserwacja, która wskazuje, że dla benchmarku OpenBookQA model BLOOM osiąga lepsze wyniki niż model Cohere xlarge niezależnie od liczby podanych przykładów w promptcie, a w przypadku benchmarku HotpotQA zachodzi odwrotna sytuacja. Dodatkowo wykorzystana strategia promptingu w pierwszym benchmarku poprawia znacznie wyniki dla modelu BLOOM, podczas gdy w kolejnym ten sam model, mimo początkowych wzrostów, osiąga ostatecznie wynik gorszy niż przy braku podanych przykładów. Tym razem to model Cohere xlarge zyskuje dużą poprawę dokładnego dopasowania. Może to być spowodowane specyfiką zbiorów pytań testowych, gdzie dla OpenBookQA ważna jest wiedza na temat prostych faktów naukowych, a dla HotPotQA sprawdzane są informacje odnośnie wielu dziedzin, które dodatkowo często wymagają porównania ich między sobą. Ograniczeniem w określeniu powodu tego zjawiska jest zdecydowanie brak informacji o zbiorze treningowym oraz sposobie uczenia Cohere xlarge. Trzeba jednak przyznać, że model ten w obu benchmarkach poprawiał swoje odpowiedzi wraz ze wzrostem liczby przykładów, co sprawia, że jest dobrym kandydatem dla wykorzystania wymienionej strategii promptingu zarówno dla pytań otwartych, jak i wielokrotnego wyboru.

Warto także zauważyć, że miary dokładnego dopasowania oraz pokrycia dokładnego dopasowania dla benchmarku OpenBookQA są do siebie na każdym etapie bardzo podobne, co świadczy o tym, że stosowana strategia wpływała głównie na treść generowanych odpowiedzi. Natomiast dla benchmarku HotpotQA początkowo miary pokrycia dokładnego dopasowania były wysokie i podobne dla obu modeli, lecz miara dokładnego dopasowania była stosunkowo mniejsza. Wraz ze

wzrostem liczby podanych przykładów model Cohere xlarge stopniowo wyrównał wyniki dla obu miar, podczas gdy BLOOM osiągał je co raz gorsze. Może to wskazywać na fakt, iż w przypadku tego benchmarku modele potrafiły odpowiedzieć na pewien zbiór pytań testowych nawet bez podania przykładów, lecz ich forma nie zgadzała się z referencyjną. Dla pierwszego modelu strategia promptingu wpłynęła głównie na doprecyzowanie generowanego tekstu, aby był on identyczny z referencyjnym, natomiast dla modelu BLOOM mogła zmienić także ich treść.

9.3 Ocena działania strategii promptingu nazwanej łańcuchem wnioskowania

Celem eksperymentu jest przetestowanie strategii promptingu zwanej łańcuchem wnioskowania (ang. chain of thought), która polega na wymuszeniu, aby model wygenerował dodatkowo pośrednie kroki udzielonej odpowiedzi. Do jego przeprowadzenia stworzyliśmy dwa wzorce dla benchmarku OpenBookQA oraz HotpotQA. Pierwszy z nich składa się z prostej instrukcji "Answer the question" i zawiera pięć przykładów odpowiedzi wraz z podanymi krokami prowadzącymi do jej udzielenia (patrz tab. A.13 w załączniku). Drugi składa się z instrukcji "Given the following question, write the step-by-step thought process and provide an answer:" i także zawiera pięć przykładów odpowiedzi wraz z przygotowanym rozumowaniem przez człowieka (patrz tab. A.15 w załączniku). Zmiennym parametrem w tym eksperymencie jest liczba podanych przykładów, która wynosi 1, 3 lub 5, w związku z czym w zależności od niej usuwaliśmy ich odpowiednią liczbę, rozpoczynając od końca promptu.

9.3.1 Opis konfiguracji systemu

Eksperiment przeprowadziliśmy dla benchmarku OpenBookQA oraz HotpotQA w dwóch różnych konfiguracjach systemu. Przetestowaliśmy wybrane modele języka, czyli BLOOM oraz Cohere xlarge, dla których ustawiliśmy parametr *temperature* na wartość 1 oraz *top_k* na wartość 0, co wpływa na większą dowolność w generowanych odpowiedziach. W tym eksperymencie nie zakładamy wyszukiwania dodatkowego kontekstu.

9.3.2 Wyniki

Wyniki dla poszczególnych miar oceny przedstawiamy poniżej. Tabele 9.16, 9.17, 9.18 dotyczą testów przeprowadzonych na benchmarku OpenBookQA dla modeli BLOOM i Cohere xlarge. Natomiast tabele 9.19, 9.20, 9.21, 9.22, 9.23, 9.24, dotyczą testów przeprowadzonych na benchmarku HotpotQA dla tych samych modeli. Dodatkowo przygotowaliśmy wizualizacje wszystkich wyników w formie wykresów.

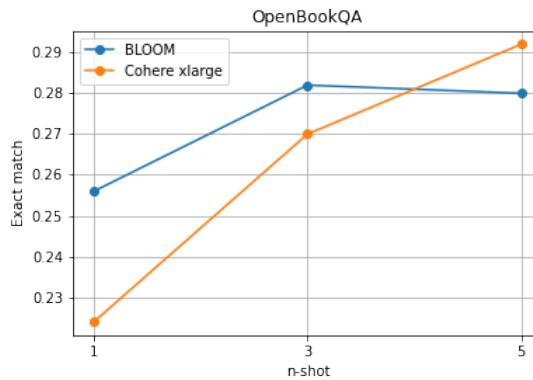
OpenBookQA

Najlepszy wynik dokładnego dopasowania osiągnęliśmy dla konfiguracji z modelem języka Cohere xlarge i wynosi on *0.292* przy zastosowaniu pięciu przykładów w promptcie (tab. 9.16). Jak wskazuje charakterystyka wykresu (rys. 9.16) w przypadku tego modelu tendencja jest wzrostowa. Różnica między podaniem jednego przykładu a podaniem pięciu przykładów wynosi *0.06*. Model BLOOM także osiąga lepsze wyniki wraz ze wzrostem liczby przykładów w promptcie, jednak maksymalna wartość dokładnego dopasowania w jego przypadku to *0.282*. Dodatkowo różnica między najgorszym i najlepszym wynikiem to tylko *0.024* a między trzema a pięcioma przykładami jest prawie niezauważalna. Warto zaznaczyć, że wykorzystując strategię łańcucha wnioskowania

z kilkoma przykładami, model Cohere xlarge mógł osiągnąć lepszy rezultat niż model BLOOM, pomimo trzykrotnie mniejszej liczby wag połączeń.

TABELA 9.16: Wartości dokładnego dopasowania w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

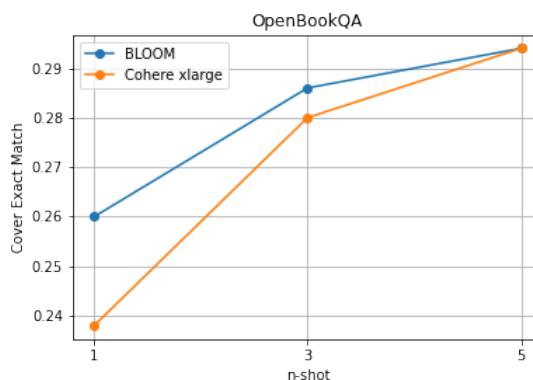
	n = 1	n = 3	n = 5
BLOOM	0.256	0.282	0.28
Cohere xlarge	0.224	0.27	0.292



RYSUNEK 9.16: Wykres zależności dokładnego dopasowania (ang. exact match) od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.17: Wartości pokrycia dokładnego dopasowania w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

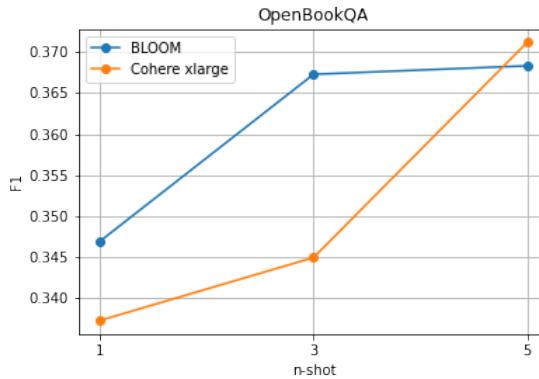
	n = 1	n = 3	n = 5
BLOOM	0.26	0.286	0.294
Cohere xlarge	0.238	0.28	0.294



RYSUNEK 9.17: Wykres zależności pokrycia dokładnego dopasowania (ang. cover exact match) od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.18: Wartości miary F1 w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

	n = 1	n = 3	n = 5
BLOOM	0.347	0.367	0.368
Cohere xlarge	0.337	0.345	0.371



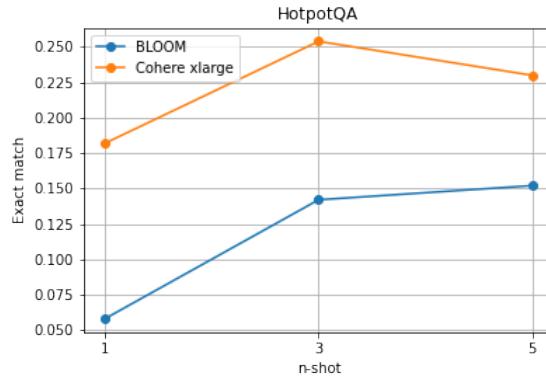
RYSUNEK 9.18: Wykres zależności miary F1 (F1 score) od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

HotpotQA

Najlepszy wynik dokładnego dopasowania osiągnęliśmy dla konfiguracji z modelem języka Cohere xlarge i wynosi on *0.254* przy zastosowaniu trzech przykładów w promptcie (tab. 9.19). Dla modelu BLOOM maksymalna wartość to *0.152* przy zastosowaniu 5 przykładów. Jak wskazują charakterystyki wykresów (rys. 9.16) w przypadku obu modeli następuje podobny wzrost wartości między jednym, a trzema podanymi przykładami. Istnieje jednak znacząca różnica. Model Cohere xlarge dla promptu z jednym przykładem osiąga wynik *0.182*, podczas gdy dokładne dopasowanie dla modelu BLOOM wynosi jedynie *0.058*. Wskazuje to na fakt, że pierwszy z nich potrafi wygenerować łańcuch wnioskowania już na podstawie jednego przykładu, a jego konkurent uczył się stopniowo wymaganej strategii i dostosowywał formę swojej odpowiedzi do instrukcji. Warto jednak wspomnieć, że nawet najlepszy wynik dokładnego dopasowania modelu BLOOM nie dorównał wartością uzyskiwanym przez model Cohere xlarge.

TABELA 9.19: Wartości dokładnego dopasowania w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

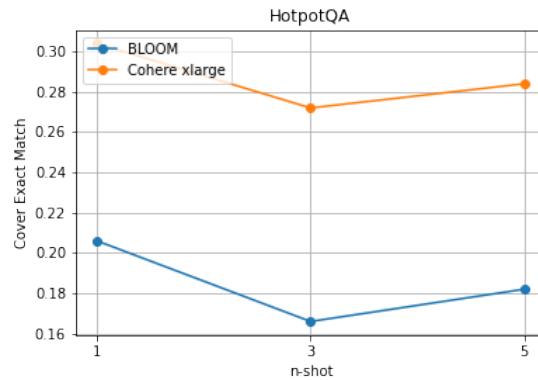
	n = 1	n = 3	n = 5
BLOOM	0.058	0.142	0.152
Cohere xlarge	0.182	0.254	0.23



RYSUNEK 9.19: Wykres zależności dokładnego dopasowania (ang. exact match) od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.20: Wartości pokrycia dokładnego dopasowania w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

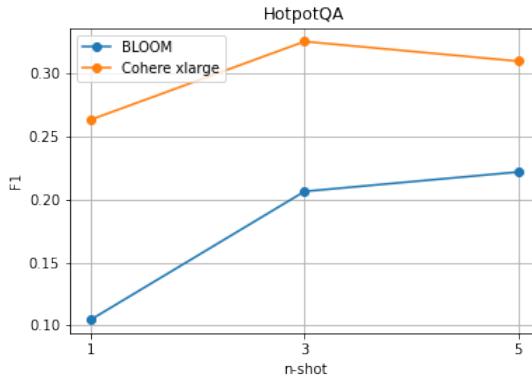
	n = 1	n = 3	n = 5
BLOOM	0.206	0.166	0.182
Cohere xlarge	0.304	0.272	0.284



RYSUNEK 9.20: Wykres zależności pokrycia dokładnego dopasowania (ang. cover exact match) od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.21: Wartości miary F1 w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

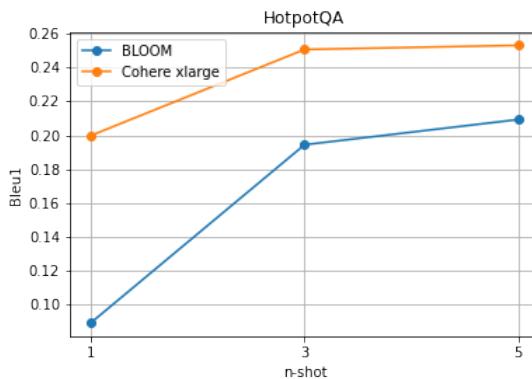
	n = 1	n = 3	n = 5
BLOOM	0.105	0.206	0.222
Cohere xlarge	0.263	0.325	0.309



RYSUNEK 9.21: Wykres zależności miary F1 (F1 score) od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.22: Wartości BLEU-1 w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

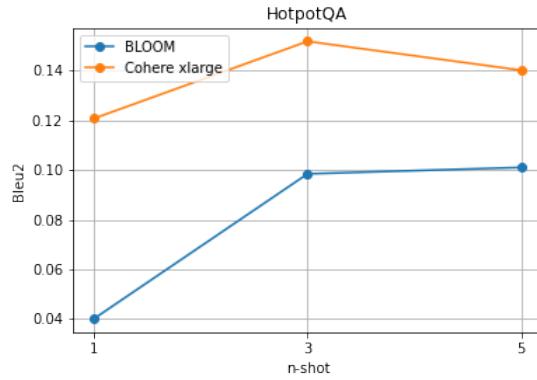
	n = 1	n = 3	n = 5
BLOOM	0.089	0.194	0.209
Cohere xlarge	0.2	0.251	0.253



RYSUNEK 9.22: Wykres zależności BLEU-1 od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.23: Wartości BLEU-2 w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

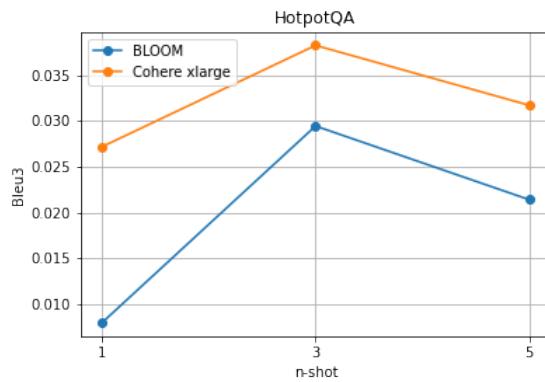
	n = 1	n = 3	n = 5
BLOOM	0.04	0.098	0.101
Cohere xlarge	0.121	0.152	0.14



RYSUNEK 9.23: Wykres zależności BLEU-2 od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.24: Wartości BLEU-3 w zależności od liczby n podanych przykładów w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

	n = 1	n = 3	n = 5
BLOOM	0.008	0.029	0.021
Cohere xlarge	0.027	0.038	0.032



RYSUNEK 9.24: Wykres zależności BLEU-3 od liczby podanych przykładów (n-shot) w promptcie z łańcuchami wnioskowania dla modelu języka BLOOM oraz Cohere xlarge.

9.3.3 Wnioski szerszej analizy wyników

Wyniki eksperymentu wskazują, że wraz ze wzrostem liczby podanych przykładów w promptcie opartym na strategii łańcucha wnioskowania, wzrastają także wyniki dla modelu Cohere xlarge oraz BLOOM. Dodatkowo dla obu benchmarków pierwszy z modeli osiągnął lepsze rezultaty. Warto jednak zwrócić uwagę, że w przypadku OpenBookQA model Cohere xlarge uzyskał przewagę nad BLOOM dopiero, przy korzystaniu z pięciu przykładów. Natomiast dla HotpotQA zdominował on całkowicie swojego rywala i od samego początku jego wyniki były znacznie lepsze, co prowadzi do wniosku, że jest on zdecydowanie atrakcyjniejszym wyborem do odpowiadania na pytania z tego benchmarku.

Interesującym okazuje się fakt, że BLOOM odpowiadając na pytania HotpotQA wykorzystując prompt z jednym przykładem, osiągnął wynik dokładnego dopasowania na poziomie zaledwie 0.058, podczas gdy Cohere xlarge od samego początku uzyskał wynik 0.182. Wskazuje to na dwie możliwości. Jedną z nich jest opcja, że model Cohere xlarge mógł po prostu szybciej dostosować generowanie odpowiedzi w strategii łańcucha wnioskowania niż BLOOM. Istnieje jednak możliwość, że jego trening był skupiony właśnie na zbiorze danych, w którym występowało dużo przykładów z testowanej w tym eksperymencie strategii, co pozwoliło, aby od samego początku umożliwiał jej wykorzystanie.

9.4 Ocena działania strategii nazwanej dynamicznym promptingiem

Celem eksperymentu jest przetestowanie wpływu stosowania strategii dynamicznego promptingu, która polega na tworzeniu indywidualnego promptu dla każdego pytania testowego na podstawie wyszukanych do niego pytań ze zbioru treningowego. Do jego przeprowadzenia przygotowaliśmy specjalny prompt z instrukcją "Answer the question based on the facts.", w którym można dynamicznie umieszczać dowolną liczbę przykładów (patrz tab. A.3 w załączniku). To właśnie ona jest zmiennym parametrem w tym eksperymencie i zawiera się w przedziale $<1,5>$. Przeprowadzony test jest testem bliźniaczym dla eksperymentu oceniającego wpływ liczby podanych przykładów rozwiązania zadania w promptcie (rozdział 9.2), lecz tym razem są one dobierane z przygotowanego zbioru treningowego według podobieństwa semantycznego do zadanego pytania. Dodatkowo wyszukiwany jest też kontekst w postaci jednego dokumentu dla każdego wybranego pytania osobno.

9.4.1 Opis konfiguracji systemu

Eksperyment przeprowadziliśmy na benchmarku OpenBookQA w dwóch różnych konfiguracjach systemu. Przetestowaliśmy wybrane modele języka, czyli BLOOM oraz Cohere xlarge, dla których ustaliliśmy parametr *temperature* na wartość 1e-6 dla pierwszego modelu i 0 dla kolejnego oraz *top_k* na wartość 1, co wpływa na większy determinizm generowanych odpowiedzi. Sprawdziliśmy ich działanie w połączeniu z metodą wyszukiwania informacji FAISS, która zapewniała zarówno wybranie pytań ze zbioru treningowego, jak i wyszukanie dla nich dokumentów stanowiących kontekst.

9.4.2 Wyniki

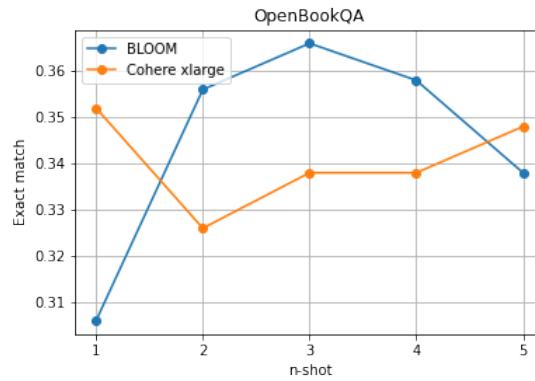
Wyniki dla poszczególnych miar oceny przedstawiamy poniżej. Tabele 9.25, 9.26, 9.27 dotyczą testów przeprowadzonych na benchmarku OpenBookQA dla modeli BLOOM i Cohere xlarge. Dodatkowo przygotowaliśmy wizualizacje wszystkich wyników w formie wykresów.

OpenbookQA

Najlepszy wynik dokładnego dopasowania osiągnęliśmy dla konfiguracji z modelem języka BLOOM i wynosi on *0.366* dla dynamicznego promptu z trzema wyszukanymi przykładami (tab. 9.25). Dla modelu Cohere xlarge maksymalny wynik jest podobny i wynosi *0.352*. Jak wskazuje charakterystyka wykresów (rys. 9.25) wartości dokładnego dopasowania dla pierwszego modelu początkowo wzrastają, lecz od wyboru trzech przykładów zaczynają maleć. W przeciwieństwie dla drugiego modelu wartość najpierw maleje, a następnie stopniowo wzrasta do początkowego poziomu.

TABELA 9.25: Wartości dokładnego dopasowania w zależności od liczby n podanych przykładów stosując dynamiczny prompting dla modelu języka BLOOM oraz Cohere xlarge.

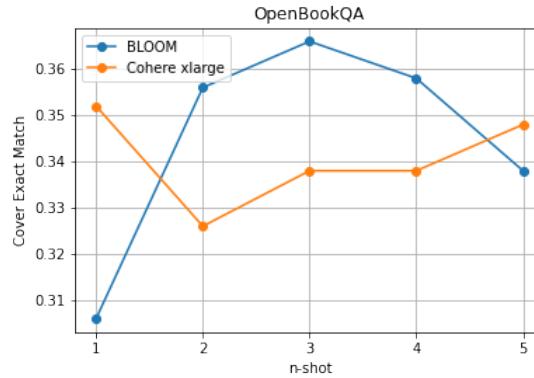
	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.306	0.356	0.366	0.358	0.338
Cohere xlarge	0.352	0.326	0.338	0.338	0.348



RYSUNEK 9.25: Wykres zależności dokładnego dopasowania (ang. exact match) od liczby podanych przykładów (n-shot) stosując dynamiczny prompting dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.26: Wartości pokrycia dokładnego dopasowania w zależności od liczby n podanych przykładów stosując dynamiczny prompting dla modelu języka BLOOM oraz Cohere xlarge.

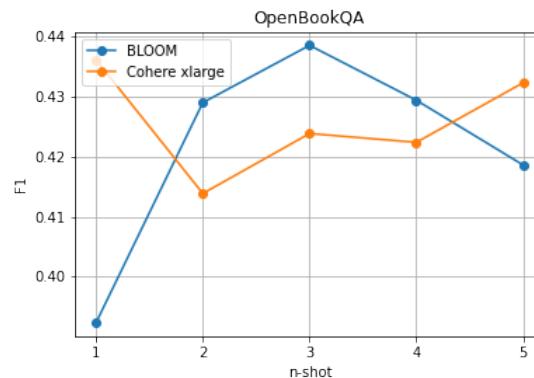
	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.306	0.356	0.366	0.358	0.338
Cohere xlarge	0.352	0.326	0.338	0.338	0.348



RYSUNEK 9.26: Wykres zależności pokrycia dokładnego dopasowania (ang. cover exact match) od liczby podanych przykładów (n-shot) stosując dynamiczny prompting dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.27: Wartości miary F1 w zależności od liczby n podanych przykładów stosując dynamiczny prompting dla modelu języka BLOOM oraz Cohere xlarge.

	n = 1	n = 2	n = 3	n = 4	n = 5
BLOOM	0.392	0.429	0.438	0.429	0.419
Cohere xlarge	0.436	0.414	0.424	0.422	0.432



RYSUNEK 9.27: Wykres zależności miary F1 (F1 score) od liczby podanych przykładów (n-shot), stosując dynamiczny prompting dla modelu języka BLOOM oraz Cohere xlarge.

9.4.3 Wnioski szerszej analizy wyników

Porównując otrzymane wyniki eksperymentu z wartościami zaobserwowanymi dla poprzedniego testu oceniającego wpływ liczby podanych przykładów na generowaną odpowiedź, można zauważyć kilka obserwacji.

Po pierwsze zastosowanie strategii dynamicznego promptingu poprawia wynik dokładnego dopasowania niezależnie od liczby wyszukiwanych przykładów z przedziału $< 1, 5 >$ w porównaniu do promptu bez podania żadnego z nich. Jak wynika z tabeli 9.7 dokładne dopasowanie wykorzystując prompt 0-shot (brak podanych przykładów) dla modelu Cohere xlarge wynosi *0.262*, a dla modelu BLOOM wynik to *0.24*. Dla porównania najniższe wyniki dla tych modeli w eksperymencie z dynamicznym promptingiem wynoszą kolejno *0.326* oraz *0.306*.

Interesującym faktem jest to, że przygotowany przez nas prompt zawierający te same przykłady dla każdego pytania testowego we wspomnianym wcześniejszym eksperymencie sprawił, iż wyniki dla modelu BLOOM okazały się o wiele wyższe niż otrzymane dla dynamicznego promptingu. Wykres 9.7 jasno wskazuje, że dokładne dopasowanie poczynając od jednego przykładu we wzorcu zawiera się w granicach *0.386 - 0.424*. Dodając tę samą liczbę przykładów wyszukanych według podobieństwa semantycznego do dynamicznego promptu otrzymaliśmy wyniki w przedziale *0.306 - 0.366*. Jednak dla modelu Cohere xlarge występuje odwrotna sytuacja. W eksperymencie ze stałym promptem wyniki na wykresie 9.7 zawierają się w granicy *0.244 - 0.306*, natomiast w opisywanym aktualnie teście jest to przedział *0.326 - 0.352*. Jak pokazują wyniki przeprowadzanych eksperymentów modele języka naturalnego są bardzo wrażliwe na użyte słownictwo w promptcie, a co za tym idzie także na wybrane przykłady rozwiązania zadania. Podobne zjawisko zaobserwowano w pracy badawczej [EL20], która wskazywała na ogromną wrażliwość na małe zmiany w instrukcjach. Dodatkowo zauważylismy, że wpływ ten nie jest reprodukowany między modelami i w zależności od stworzonego promptu może się powodować inną zmianę wyników w różnych wariantach modeli. Podobne wnioski stwierdzono w trakcie badań nad kolejnością permutacji przykładów w promptcie, gdzie osiągający wysokie wyniki prompt na jednym modelu, drastycznie zaniża je na innym [LBM⁺21a].

9.5 Ocena wpływu kolejności występowania przykładów w promptcie

Celem eksperymentu jest przetestowanie wpływu kolejności występowania przykładów na wyniki generowane przez modele języka. Do jego przeprowadzenia przygotowaliśmy wzorzec dla benchmarku OpenBookQA, który składa się z prostej instrukcji "Answer the question based on the facts." oraz pięciu przykładów z podanym jednym dokumentem jako fakt, których kolejność była zmieniana dla każdego etapu testu (patrz tab. A.12 w załączniku). Zakładając ich numerację według kolejności w zaproponowanym propmcie zaczynając od liczby 1, przetestowaliśmy zachowanie modelu dla następujących permutacji:

- **I** - 1, 2, 3, 4, 5
- **II** - 5, 1, 4, 2, 3
- **III** - 3, 5, 2, 4, 1
- **IV** - 5, 4, 3, 2, 1
- **V** - 1, 5, 3, 4, 2
- **VI** - 4, 5, 1, 2, 3

9.5.1 Opis konfiguracji systemu

Eksperyment przeprowadziliśmy na benchmarku OpenBookQA w dwóch różnych konfiguracjach systemu. Przetestowaliśmy wybrane modele języka, czyli BLOOM oraz Cohere xlarge, dla

których ustawiliśmy parametr *temperature* na wartość 1e-6 dla pierwszego modelu i 0 dla kolejnego oraz *top_k* na wartość 1, co wpływa na większy determinizm generowanych odpowiedzi. Sprawdziliśmy ich działanie w połączeniu z metodą wyszukiwania informacji FAISS.

9.5.2 Wyniki

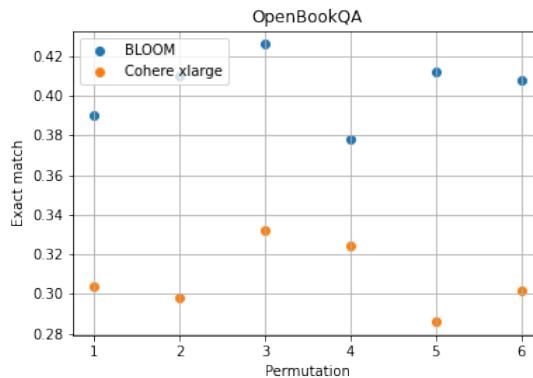
Wyniki dla poszczególnych miar oceny przedstawiamy poniżej. Tabele 9.28, 9.29, 9.30 dotyczą testów przeprowadzonych na benchmarku OpenBookQA dla modeli BLOOM i Cohere xlarge. Dodatkowo przygotowaliśmy wizualizacje wszystkich wyników w formie wykresów.

OpenBookQA

Najlepszy wynik dokładnego dopasowania osiągnęliśmy dla konfiguracji z modelem języka BLOOM i wynosi on *0.426* dla permutacji III (tab. 9.28). Dla modelu Cohere xlarge maksymalny wynik wystąpił także dla permutacji III i wynosi *0.332*. Jak wskazują wartości na wykresie (rys. 9.28) w zależności od wybranej kolejności przykładów w promptcie osiągane wyniki dla każdego z modeli różnią się. W przypadku pierwszego z nich różnica między najlepszym i najgorszym dokładnym dopasowaniem to *0.048*, a dla kolejnego wynosi ona niemal identycznie *0.046*. Należy także zwrócić uwagę, że pomimo zgody co do najlepszej z zaproponowanych kolejności przykładów, kolejne z nich nie przedstawiają podobnych zależności.

TABELA 9.28: Wartości dokładnego dopasowania w zależności od wybranej permutacji przykładów w promptcie dla modelu języka BLOOM oraz Cohere xlarge.

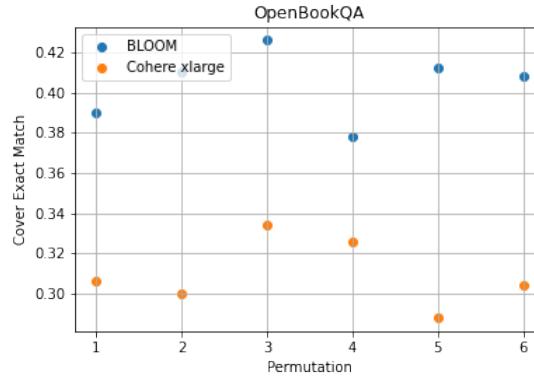
	I	II	III	IV	V	VI
BLOOM	0.39	0.41	0.423	0.378	0.412	0.408
Cohere xlarge	0.304	0.298	0.332	0.324	0.286	0.302



RYSUNEK 9.28: Wykres zależności dokładnego dopasowania (ang. exact match) od wybranej permutacji przykładów w promptcie dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.29: Wartości pokrycia dokładnego dopasowania w zależności od wybranej permutacji przykładów w promptcie dla modelu języka BLOOM oraz Cohere xlarge.

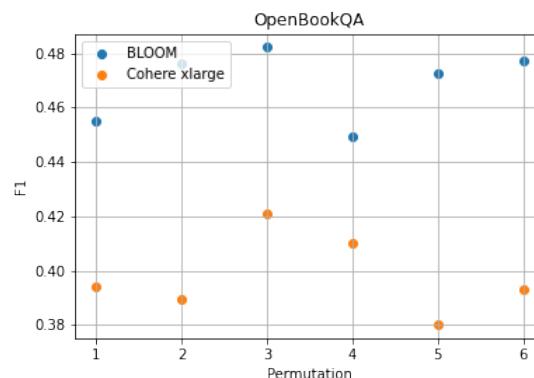
	I	II	III	IV	V	VI
BLOOM	0.39	0.41	0.423	0.378	0.412	0.408
Cohere xlarge	0.306	0.3	0.334	0.326	0.288	0.304



RYSUNEK 9.29: Wykres zależności pokrycia dokładnego dopasowania (ang. cover exact match) od wybranej permutacji przykładów w promptcie dla modelu języka BLOOM oraz Cohere xlarge.

TABELA 9.30: Wartości miary F1 w zależności od wybranej permutacji przykładów w promptcie dla modelu języka BLOOM oraz Cohere xlarge.

	I	II	III	IV	V	VI
BLOOM	0.455	0.476	0.482	0.449	0.473	0.477
Cohere xlarge	0.394	0.389	0.421	0.41	0.38	0.393



RYSUNEK 9.30: Wykres zależności miary F1 (F1 score) od wybranej permutacji przykładów w promptcie dla modelu języka BLOOM oraz Cohere xlarge.

9.5.3 Wnioski szerszej analizy wyników

Przeprowadzony eksperyment wskazuje, że generowane odpowiedzi przez modele języka są wrażliwe na zastosowaną kolejność przykładów w promptcie. Potwierdzają to wcześniej przeprowadzane badania, w których autorzy zauważali, że permutacje mają ogromne znaczenie na jakość osiąganych wyników i istnieje możliwość znalezienia najlepszej z nich [LBM⁺21b]. Dodatkowo zauważaliśmy, że

ranking permutacji tych samych przykładów w promptcie nie jest identyczny dla każdego modelu. Oznacza to, że najgorszy z promptów dla jednego z nich nie musi wcale generować najgorszych wyników dla drugiego.

Warto również wspomnieć, że różnice dokładnego dopasowania między najlepszą a najgorszą permutacją dla BLOOM i Cohere xlarge to około 5%. Podobne zależności zauważono w trakcie testowania wpływu kolejności przykładów w promptcie dla modelu GPT-3, gdzie poszczególne wartości znajdowały się w granicach 54.3% - 93.4% [ZWF⁺²¹].

9.6 Ogólne wnioski przeprowadzonych eksperymentów

Wyniki przeprowadzonych eksperymentów wyraźnie wskazują, że w zależności od użytej strategii promptingu modele języka BLOOM oraz Cohere xlarge generują inne odpowiedzi.

- Dla benchmarku OpenBookQA:
 - Najlepszy wynik dokładnego dopasowania wykorzystując model BLOOM osiągnęliśmy podając prompt z podstawową instrukcją oraz dwoma przykładami z jednym wyszukanym dokumentem i wynosi on *0.424* (tab. 9.7).
 - W przypadku konfiguracji z modelem Cohere xlarge najlepszy okazał się dynamiczny prompting z jednym wyszukanym przykładem i jednym dokumentem, dla którego wartość dokładnego dopasowania to *0.352* (tab. 9.10).
- Dla benchmarku HotpotQA:
 - Najwyższą wartość dokładnego dopasowania *0.254* osiągnął model Cohere xlarge wykorzystując strategię łańcucha wnioskowania (tab. 9.19).
 - Natomiast model BLOOM zdobył maksymalny wynik *0.164* dla promptu z podstawową instrukcją i dwoma przykładami z wyszukiwaniem jednego dokumentu (tab. 9.10).

Na podstawie powyższych wyników można określić, że dla benchmarku OpenBookQA najlepszy okazał się model BLOOM, a dla HotpotQA model Cohere xlarge.

Warto nadmienić, że przetestowane strategie promptingu oraz wykorzystane prompty zazwyczaj wpływały pozytywnie na generowane przez modele odpowiedzi. Uznając za bazowe wyniki wartości dokładnego dopasowania w przypadku promptu z podstawową instrukcją oraz braku podanych przykładów, które wynoszą:

- Dla benchmarku OpenBookQA (tab. 9.7):
 - *0.24* w przypadku modelu BLOOM
 - *0.262* w przypadku modelu Cohere xlarge
- Dla benchmarku HotpotQA (tab. 9.10):
 - *0.1* w przypadku modelu BLOOM
 - *0.094* w przypadku modelu Cohere xlarge

, można zauważyć, że odpowiednio dobrane parametry w postaci liczby wyszukanych dokumentów lub liczby podanych przykładów pozwalają na poprawę generowanych odpowiedzi. Należy mieć jednak na uwadze, iż wyniki różnią się znacznie pomiędzy strategiami promptingu i należy je dobrać zarówno do modelu, jak i danego benchmarku.

Zaobserwowałyśmy także, że model BLOOM oraz Cohere xlarge są wrażliwe na kolejność przykładów w promptcie. W naszym eksperymencie dla benchmarku OpenBookQA różnice dokładnego dopasowania wynosiły około *0.05* (tab. 9.28), co uważamy za znaczącą wielkość, jeśli porównamy ją do różnic między bazowym, a najlepszym osiągniętym wynikiem ze wszystkich strategii:

- Różnica dla modelu BLOOM wynosi *0.184*
- Różnica dla modelu Cohere xlarge wynosi *0.09*

Dodatkowo musimy wspomnieć, że przetestowałyśmy zaledwie sześć permutacji. Oznacza to, iż poszukując odpowiedniej strategii promptingu należy mieć także na uwadze kolejność podawania przykładów, co w ostatecznym rozrachunku zwiększa przestrzeń promptów do przeszukiwania.

Podobną wrażliwość, lecz tym razem związaną z występowaniem słów w promptcie, zauważałyśmy przy stosowaniu dynamicznego promptingu. Wyszukiwane przykładów dla każdego pytania testowego doprowadziło do uzyskania odmiennych wyników niż z wykorzystaniem stałego promptu. Dodatkowo model Cohere xlarge osiągał lepsze wyniki dokładnego dopasowania z dynamicznym promptem, a model BLOOM ze stałym.

- Wyniki dokładnego dopasowania dla modelu BLOOM ze stałym promptem zawierają się w przedziale *0.386 - 0.424* (tab. 9.7), natomiast dla dynamicznego promptu w przedziale *0.306 - 0.366* (tab. 9.25)
- Wyniki dokładnego dopasowania dla modelu Cohere xlarge ze stałym promptem zawierają się w przedziale *0.244 - 0.306* (tab. 9.7), natomiast dla dynamicznego promptu w przedziale *0.326 - 0.352* (tab. 9.25)

Oznacza to, że stosowane prompty mimo świetnych wyników dla jednego z modeli nie gwarantują identycznego zachowania dla innego. Wskazuje to ponownie na potrzebę dostosowywania wybieranej strategii i promptów do określonego modelu.

Zaobserwowałyśmy również, iż wraz ze wzrostem liczby podanych przykładów modele BLOOM oraz Cohere potrafili zaadaptować się do zadania i dostosować swoje odpowiedzi, co doprowadziło do uzyskania lepszych wyników dokładnego dopasowania w porównaniu do podejścia 0-shot zarówno dla promptu z podstawową instrukcją jak i podanymi przykładami (tab. 9.7, 9.10), promptu korzystającego z łańcucha wnioskowania (tab. 9.16, 9.19), a także dynamicznego promptingu (tab. 9.25). Należy jednak zwrócić uwagę na charakterystyki wykresów (rys. 9.7, 9.10, 9.19, 9.16, 9.25), które wskazują, że trend nie jest zawsze wzrostowy i zazwyczaj po osiągniętym maksimum następuje spadek lub utrzymanie na stałym poziomie.

Chcielibyśmy także zauważać, że określenie niektórych zachowań modeli BLOOM oraz Cohere xlarge jest trudne w związku z wykorzystaniem zewnętrznych API, nad którymi nie posiadamy pełnej kontroli. Osiągnięte wyniki mogą także różnić się podczas ponownego odtworzenia eksperymentów, jeśli twórcy udostępniający swoje modele postanowią wprowadzić znaczne zmiany.

Podsumowując przeprowadzone eksperymenty stwierdzamy, że wykorzystanie strategii promptingu może wpływać pozytywnie na wyniki osiągane przez modele języka naturalnego. Pomimo tego wymagają one odpowiedniego dobrania oraz przygotowania specjalnego promptu w zależności od modelu, a także charakterystyki pytań. Uniemożliwia to określenie uniwersalnego rozwiązania, które sprawdzi się w każdym przypadku.

Rozdział 10

Przykłady użycia

Efektem naszej pracy jest stworzona biblioteka w języku python, a także skonfigurowany na jej podstawie system udzielający odpowiedzi na pytanie z wyszukiwaniem kontekstu. Co więcej, dzięki przygotowanej aplikacji internetowej z interfejsem użytkownika, możliwe jest proste odpytanie wcześniej przygotowanego rozwiązania.

W tym rozdziale prezentujemy przykłady użycia wymienionej wcześniej biblioteki, a także proces konfiguracji systemu wraz z wygenerowanymi odpowiedziami na wybrane pytania.

10.1 Użycie biblioteki

Biblioteka składa się z opisanych we wcześniejszym rozdziale modułów, z których za bazowe części można uznać moduł *information_retrievers* pozwalający wyszukać dokumenty w wybranych zbiorach danych, a także moduł *language_models*, który umożliwia odpytywanie modeli języka naturalnego. W celu użycia niektórych z komponentów należy uprzednio ustawić odpowiednie zmienne środowiskowe zawierające klucze użytkownika wymagane przez zewnętrzne API, przykładowo dla modelu Cohere xlarge. Szczegóły oraz instrukcje ich uzyskania znajdują się w utworzonym repozytorium kodu¹. Po przygotowaniu środowiska oraz zimportowaniu biblioteki możliwe jest skorzystanie zarówno z poszczególnych komponentów osobno, jak i w zestawieniu w postaci klasy *PIQARD*, lub zaimplementowanych, rozszerzonych strategii promptingu *ReAct* oraz *SelfAware*. Poniżej umieściliśmy przykłady w środowisku jupyter notebook.

¹<https://github.com/plaskod/piqard>

- Odpytanie modelu języka Cohere xlarge z przygotowanym promptem:

```

1 from piqard.utils.prompt_template import PromptTemplate
2 from piqard.language_models import CohereAPI
3
4 prompt_template_string = """
5 Answer the question.
6 Q: {question}
7 A:
8 """
9
10 prompt_template = PromptTemplate(prompt_template_string)
11 cohere = CohereAPI(stop_token="\n")
12
13 cohere.query(prompt_template.render(question="How long do african elephants live?"))

'African elephants live up to 70 years.'

```

RYSUNEK 10.1: Odpytanie modelu języka Cohere xlarge z przygotowanym promptem.

- Wyszukanie dokumentu z przygotowanego zbioru za pomocą metody FAISS:

```

1 from piqard.information_retrievers import FAISSRetriever
2
3 faiss = FAISSRetriever(k=5,
4                         database="openbookqa",
5                         database_path="../assets/benchmarks/openbookqa/corpus.jsonl",
6                         database_index="../assets/benchmarks/openbookqa/corpus_faiss_index.pickle")
7
8 faiss.get_documents("The sun is responsible for")

corpus.jsonl: 100% [██████████] 1326/1326 [00:00<00:00, 121232.17it/s]

['the sun is the source of solar energy called sunlight',
 'the sun is a source of heat called sunlight',
 'the sun is a source of light called sunlight',
 'the sun is the source of energy for life on Earth',
 'the sun is the source of energy for physical cycles on Earth']

```

RYSUNEK 10.2: Wyszukanie dokumentu z przygotowanego zbioru za pomocą metody FAISS.

- Wykorzystanie klasy PIQARD z modelem BLOOM, wyszukiwaniem informacji za pomocą AnnoyRetriever oraz przygotowanym wcześniej plikiem tekstowym zawierającym prompt w strategii łańcucha wnioskowania:

```

1 from piqard.utils.prompt_template import PromptTemplate
2 from piqard.information_retrievers import AnnoyRetriever
3 from piqard.language_models import BLOOM176bAPI
4 from piqard.PIQARD import PIQARD
5
6 prompt_template = PromptTemplate(template="../../assets/prompting_templates/openbookqa/chain_of_thought/cot_5_shot.txt",
7                                     fix_text="So the final answer is:")
8 bloom = BLOOM176bAPI(stop_token="|||")
9 annoy = AnnoyRetriever(k=5,
10                        database="openbookqa",
11                        database_path="../../assets/benchmarks/openbookqa/corpus.jsonl",
12                        database_index="../../assets/benchmarks/openbookqa/corpus_annoy_index_384.ann")
13
14 piqard = PIQARD(prompt_template=prompt_template,
15                  language_model=bloom,
16                  information_retriever=annoy)
17
18 question = "There is most likely going to be fog around?"
19 possible_answers = "A. a marsh B. a tundra C. the plains D. a desert"
20
21 result = piqard(query=question, possible_answers=possible_answers)
22 print(result["chain_trace"])

```

corpus.jsonl: 100% 1326/1326 [00:00<00:00, 113082.98it/s]

[observation] fog is formed by water vapor condensing in the air grey clouds appear before precipitation bad weather decreases visibility while driving a warm front causes cloudy and rainy weather storms can cause a landslide
[base_prompt] Answer the question

Question: The sun is responsible for
Possible answers: A. puppies learning new tricks, B. children growing up and getting old, C. flowers wilting in a vase, D. plants sprouting, blooming and wilting
Let's think step by step:

- puppies can learn new tricks even in the dark or inside a house
- children growing up and getting old is a natural cycle of life
- flowers wilting in a vase could result from not supplying enough water
- sunlight is essential for plants for photosynthesis which make them grow, sprout, bloom and wilt when there's not enough of it
So the final answer is: D. plants sprouting, blooming and wilting
|||

• • •

Question: Which is most harmful to a human?
Possible answers: A. dynamite going off, B. reading email, C. balloons popping, D. opening a soda
Let's think step by step:
- a person may get killed if they're too close of a dynamite explosion, the shockwave can also damage hearing
- reading an email is not dangerous, at worst the email contains a computer virus which does not directly impact human's health
- balloons popping can be very irritating, but do not lead to hearing loss
- it's very unlikely to be harmed while opening a soda, at worst you cut your finger on a piece of aluminium
So the final answer is: A. dynamite going off
|||

Question: There is most likely going to be fog around?
Possible answers: A. a marsh B. a tundra C. the plains D. a desert
Let's think step by step:

[thought]
- a marsh is a wetland area where there is a lot of water
- a tundra is a cold area with a lot of snow
- the plains are a flat area with a lot of grass
- a desert is a dry area with a lot of sand
- fog is a cloud of water droplets
So the final answer is: A. a marsh

[finish] A. a marsh

RYSUNEK 10.3: Wykorzystanie klasy PIQARD z modelem BLOOM, wyszukiwaniem informacji za pomocą AnnoyRetriever oraz przygotowanym wcześniej plikiem tekstowym zawierającym prompt w strategii łańcucha wnioskowania.

- Wykorzystanie klasy Agent z modelem Cohere, wyszukiwaniem informacji za pomocą WikiAPI oraz przygotowanym wcześniej plikiem tekstowym zawierającym prompt w strategii ReAct:

```

1 from piqard.utils.prompt_template import PromptTemplate
2 from piqard.information_retrievers import WikiAPI
3 from piqard.language_models import CohereAPI
4 from piqard.extensions.react import Agent, Action
5
6 prompt_template = PromptTemplate(template='./assets/prompting_templates/react/react_prompt.txt')
7 cohore = CohereAPI(stop_token="\n", temperature=1, top_k=0)
8 wiki = WikiAPI(k=10)
9
10 actions = [
11     Action("Wiki", wiki, prefix="Search")
12 ]
13
14 react_agent = Agent(prompt_template=prompt_template,
15                      language_model=cohore,
16                      actions=actions)
17
18 result = react_agent("Was Queen Elizabeth II buried in Pantheon, Rome?")
19 print(result['chain_trace'])

```

[base_prompt] Question: What profession does Nicholas Ray and Elia Kazan have in common?
Thought 1: I need to search Nicholas Ray and Elia Kazan, find their professions, then find the profession they have in common.
Action 1: Search[Nicholas Ray]
Observation: Nicholas Ray (born Raymond Nicholas Kienzle Jr., August 7, 1911 – June 16, 1979) was an American film director, screenwriter, and actor best known for the 1955 film *Rebel Without a Cause*.
Thought 2: Professions of Nicholas Ray are director, screenwriter, and actor. I need to search Elia Kazan next and find his professions.
Action 2: Search[Elia Kazan]
Observation: Elia Kazan was an American film and theatre director, producer, screenwriter and actor.
Thought 3: Professions of Elia Kazan are director, producer, screenwriter, and actor. So profession Nicholas Ray and Elia Kazan have in common is director, screenwriter, and actor.
Action 3: Finish[director, screenwriter, actor]

● ● ●

Question: Was Queen Elizabeth II buried in Pantheon, Rome?
[thought] Thought 1: I need to search Pantheon, Rome and Queen Elizabeth II, and find if she was buried in Pantheon, Rome.
[action] Action 1: Search[Pantheon, Rome]
[observation] Observation: The Pantheon (UK: , US: ; Latin: Pantheum, from Greek Πάνθεον Pantheion, "[temple] of all the gods") is a former Roman temple and, since 609 AD, a Catholic church (Basilica di Santa Maria ad Martyres or Basilica of St. Mary and the Martyrs) in Rome, Italy, on the site of an earlier temple commissioned by Marcus Agrippa during the reign of Augustus (27 BC – 14 AD). It was rebuilt by the emperor Hadrian and probably dedicated c. 126 AD. Its date of construction is uncertain, because Hadrian chose not to inscribe the new temple but rather to retain the inscription of Agrippa's older temple, which had burned down. The building is cylindrical with a portico of large granite Corinthian columns (eight in the first rank and two groups of four behind) under a pediment. A rectangular vestibule links the porch to the rotunda, which is under a coffered concrete dome, with a central opening (oculus) to the sky. Almost two thousand years after it was built, the Pantheon's dome is still the world's largest unreinforced concrete dome. The height to the oculus and the diameter of the interior circle are the same, 43 metres (142 ft). It is one of the best-preserved of all Ancient Roman buildings, in large part because it has been in continuous use throughout its history: since the 7th century, it has been a church dedicated to St. Mary and the Martyrs (Latin: Sancta Maria ad Martyres) but informally known as "Santa Maria Rotonda". The square in front of the Pantheon is called Piazza della Rotonda. The Pantheon is a state property, managed by Italy's Ministry of Cultural Heritage and Activities and Tourism through the Polo Museale del Lazio.
[thought] Thought 2: I need to search Queen Elizabeth II next and find if she was buried in Pantheon, Rome.
[action] Action 2: Search[Queen Elizabeth II]
[observation] Observation: Elizabeth II (Elizabeth Alexandra Mary; 21 April 1926 – 8 September 2022) was Queen of the United Kingdom and other Commonwealth realms from 6 February 1952 until her death in 2022. She was queen regnant of 32 sovereign states during her lifetime, and was head of state of 15 realms at the time of her death. Her reign of 70 years and 214 days was the longest of any British monarch and the longest verified reign of any female monarch in history. Elizabeth was born in Mayfair, London, as the first child of the Duke and Duchess of York (later King George VI and Queen Elizabeth The Queen Mother). Her father acceded to the throne in 1936 upon the abdication of his brother Edward VIII, making the ten-year-old Princess Elizabeth the heir presumptive. She was educated privately at home and began to undertake public duties during the Second World War, serving in the Auxiliary Territorial Service. In November 1947, she married Philip Mountbatten, a former prince of Greece and Denmark, and their marriage lasted 73 years until his death in 2021. They had four children: Charles, Anne, Andrew, and Edward. When her father died in February 1952, Elizabeth—then 25 years old—became queen of seven independent Commonwealth countries: the United Kingdom, Canada, Australia, New Zealand, South Africa, Pakistan, and Ceylon (known today as Sri Lanka), as well as head of the Commonwealth. Elizabeth reigned as a constitutional monarch through major political changes such as the Troubles in Northern Ireland, devolution in the United Kingdom, the decolonisation of Africa, and the United Kingdom's accession to the European Communities and withdrawal from the European Union.
[thought] Thought 3: She was not buried in Pantheon, Rome.
[action] Action 3: Finish[not]
[finish] not

RYSUNEK 10.4: Wykorzystanie klasy Agent z modelem Cohere, wyszukiwaniem informacji za pomocą WikiAPI oraz przygotowanym wcześniej plikiem tekstowym zawierającym prompt w strategii ReAct.

- Wykorzystanie klasy `SelfAware` z modelem `Cohere`, która została zainicjowana obiektem klasy `PIQARD` z promptem bez wyszukiwania informacji oraz obiektem klasy `PIQARD` z wyszukiwaniem informacji za pomocą `WikiAPI` oraz przygotowanym wcześniej plikiem tekstowym zawierającym prompt podobny do chatGPT:

```

1 from piqard.utils.prompt_template import PromptTemplate
2 from piqard.information_retrievers import WikiAPI
3 from piqard.language_models import CohereAPI
4 from piqard.PIQARD import PIQARD
5 from piqard.extensions.self_aware import SelfAware
6
7 # if should not browse
8 should_not_browse_prompt_template = PromptTemplate('./../assets/prompting_templates/like_chat_gpt/without_context.txt')
9 should_not_browse_llm = CohereAPI(stop_token="\n")
10
11 should_not_browse = PIQARD(prompt_template=should_not_browse_prompt_template,
12                             language_model=should_not_browse_llm,
13                             information_retriever=None)
14
15 # if should browse
16 should_browse_prompt_template = PromptTemplate('./../assets/prompting_templates/like_chat_gpt/without_context.txt')
17 should_browse_llm = CohereAPI(stop_token="\n")
18 should_browse_ir = WikiAPI(k=10)
19
20 should_browse = PIQARD(prompt_template=should_browse_prompt_template,
21                         language_model=should_browse_llm,
22                         information_retriever=should_browse_ir)
23
24 # selfAware
25 self_aware_prompt_template = PromptTemplate('./../assets/prompting_templates/self_aware/self_aware_prompt.txt')
26 self_aware_llm = CohereAPI(stop_token="\n")
27
28 self_aware = SelfAware(prompt_template=self_aware_prompt_template,
29                         language_model=self_aware_llm,
30                         if_should_browse=should_browse,
31                         if_should_not_browse=should_not_browse)
32
33 result = self_aware("What is yellow color?")
34 print(result['chain_trace'])

```

[thought] Should I browse the web for an answer?: no
[base_prompt] Assistant is a large language model.

Assistant is designed to be able to assist with a wide range of tasks, from answering simple questions to providing in-depth explanations and discussions on a wide range of topics. As a language model, Assistant is able to generate human-like text based on the input it receives, allowing it to engage in natural-sounding conversations and provide responses that are coherent and relevant to the topic at hand.

Assistant is constantly learning and improving, and its capabilities are constantly evolving. It is able to process and understand large amounts of text, and can use this knowledge to provide accurate and informative responses to a wide range of questions. Additionally, Assistant is able to generate its own text based on the input it receives, allowing it to engage in discussions and provide explanations and descriptions on a wide range of topics.

Overall, Assistant is a powerful tool that can help with a wide range of tasks and provide valuable insights and information on a wide range of topics. Whether you need help with a specific question or just want to have a conversation about a particular topic, Assistant is here to assist.

Human: What is yellow color?
Assistant:
[thought] Yellow is a color that is often associated with happiness and optimism. It is also the color of the sun, which is often seen as a symbol of hope and new beginnings.
[finish] Yellow is a color that is often associated with happiness and optimism. It is also the color of the sun, which is often seen as a symbol of hope and new beginnings.

RYSUNEK 10.5: Wykorzystanie klasy `SelfAware` z modelem `Cohere`, która została zainicjowana obiektem klasy `PIQARD` z promptem bez wyszukiwania informacji oraz obiektem klasy `PIQARD` z wyszukiwaniem informacji za pomocą `WikiAPI` oraz przygotowanym wcześniej plikiem tekstowym zawierającym prompt podobny do chatGPT.

10.2 Użycie systemu

W celu umożliwienia użytkownikowi prostej interakcji z systemem stworzyliśmy aplikację internetową z interfejsem graficznym oraz zaproponowaliśmy jego konfigurację. Podobnie jak w przypadku użycia biblioteki wymagane jest ustalenie odpowiednich kluczy użytkownika, lecz tym razem dzieje się to w pliku konfiguracyjnym API o nazwie *config.json*. Dodatkowo należy umieścić w nim ścieżkę do pliku .yaml zawierającego przygotowaną konfigurację systemu, a następnie uruchomić serwer wywołując komendę *python piqard_api/api.py* w folderze głównym projektu. Jedno z proponowanych przez nas ustawień znajduje się na rys. 10.6. Kolejnym krokiem jest uruchomienie aplikacji internetowej poprzez wywołanie komendy *npm start* w folderze *pigard_app*. Użytkownik automatycznie zostanie przekierowany do przeglądarki z widokiem strony głównej. Dokładna instrukcja użycia systemu znajduje się w repozytorium naszego projektu². Poniżej przedstawiamy odpowiedzi na wybrane pytania wygenerowane przez zaproponowany system w konfiguracji z rys. 10.6 oraz zalety wynikające z jego wykorzystania.

```
prompt_template: !PromptTemplate
  template: assets/prompting_templates/self_aware/self_aware_prompt.txt
language_model: !CohereAPI
  stop_token: "\n"
  temperature: 0
  top_k: 1
if_should_not_browse: !PIQARD
  prompt_template: !PromptTemplate
    template: assets/prompting_templates/like_chat_gpt/without_context.txt
language_model: !CohereAPI
  stop_token: "\n"
if_should_browse: !Agent
  prompt_template: !PromptTemplate
    template: assets/prompting_templates/react/react_prompt.txt
language_model: !CohereAPI
  stop_token: "\n"
  temperature: 0.5
  top_k: 0
actions:
  - !Action
    name: Wikipedia
    func: !WikiAPI
    k: 10
    prefix: Search
```

RYSUNEK 10.6: Przykład pliku .yaml z konfiguracją systemu.

²<https://github.com/plaskod/piqard>

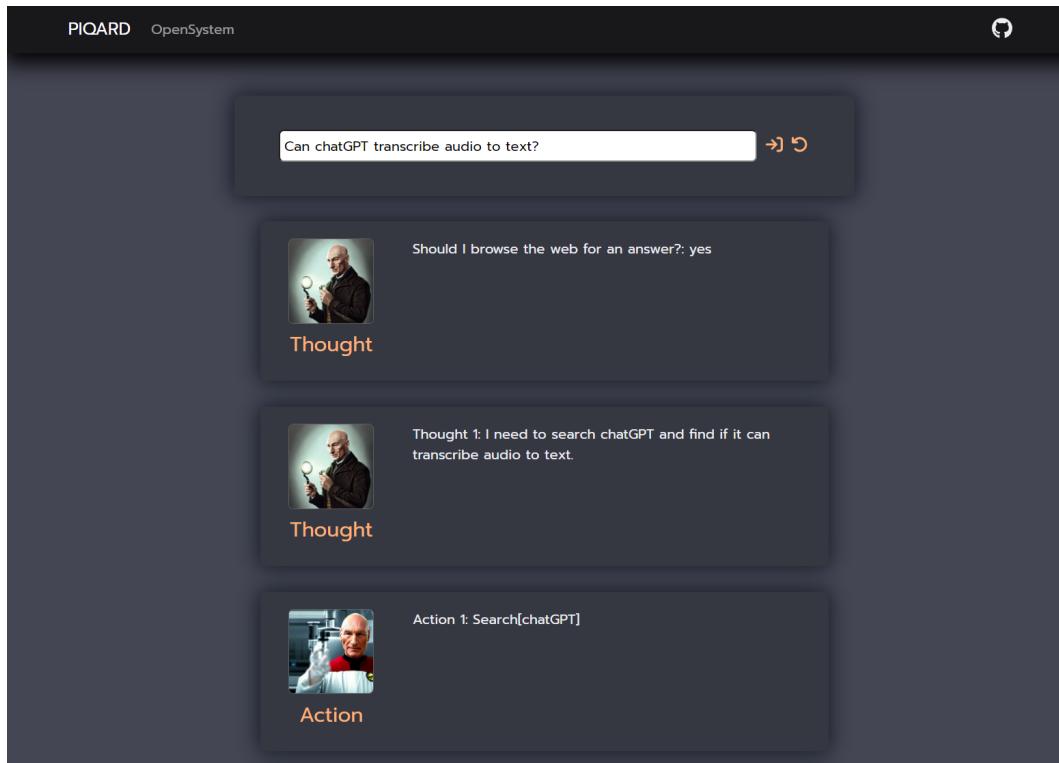
- Pytanie: Can chatGPT transcribe audio to text?

Zaproponowany system udzielił poprawną odpowiedź oraz podał źródło, z którego skorzystał, aby zaktualizować swoją wiedzę (rys. 10.8, 10.9). Natomiast odpowiedź wygenerowana przez proste odpytanie modelu języka jest niepoprawna (rys. 10.7). Na dodatek niezależnie, czy będzie ona brzmieć pozytywnie, czy negatywnie nie ma żadnych podstaw, gdyż zbiór treninowy nie mógł zawierać informacji o chatbocie chatGPT³, który powstał po wytrenowaniu modelu (30.11.2022 r.).

```
1 basic_prompt_template = """Answer the question.
2 Q: {{question}}
3 A:
4 """
5 basic_prompt = PromptTemplate(template=basic_prompt_template)
6 cohere = CohereAPI(stop_token="\n")
7
8 cohere.query(basic_prompt.render(question="Can chatGPT transcribe audio to text?"))
```

'Yes, chatGPT can transcribe audio to text.'

RYSUNEK 10.7: Odpowiedź wygenerowana przez proste zapytanie modelu języka.



RYSUNEK 10.8: Odpowiedź wygenerowana przez system (część 1/2).

³<https://openai.com/blog/chatgpt/>

The screenshot shows a conversational interface with four main sections:

- Observation:** A portrait of a man in a historical-style coat, followed by text about ChatGPT's history and development.
- Thought:** A portrait of a man holding a magnifying glass, followed by the text "Thought 2: It looks like chatGPT is a chatbot and can not transcribe audio to text."
- Action:** A portrait of a man in a white lab coat, followed by the text "Action 2: Finish[no]".
- Answer:** A portrait of a man in a white lab coat, followed by the word "no".

At the bottom, the text "PIQARD Open System" and "Try again ⌂" are visible.

RYSUNEK 10.9: Odpowiedź wygenerowana przez system (część 2/2).

- **Pytanie: Who is the current CEO of twitter?**

Zaproponowany system udzielił poprawną odpowiedź oraz podał źródło, z którego skorzystał, aby zaktualizować swoją wiedzę (rys. 10.11, 10.12). Natomiast odpowiedź wygenerowana przez proste odpytanie modelu języka jest niepoprawna (rys. 10.10). Model wspomina o byłym szefie Twittera, który został zastąpiony oficjalnie 27 października 2022 r.

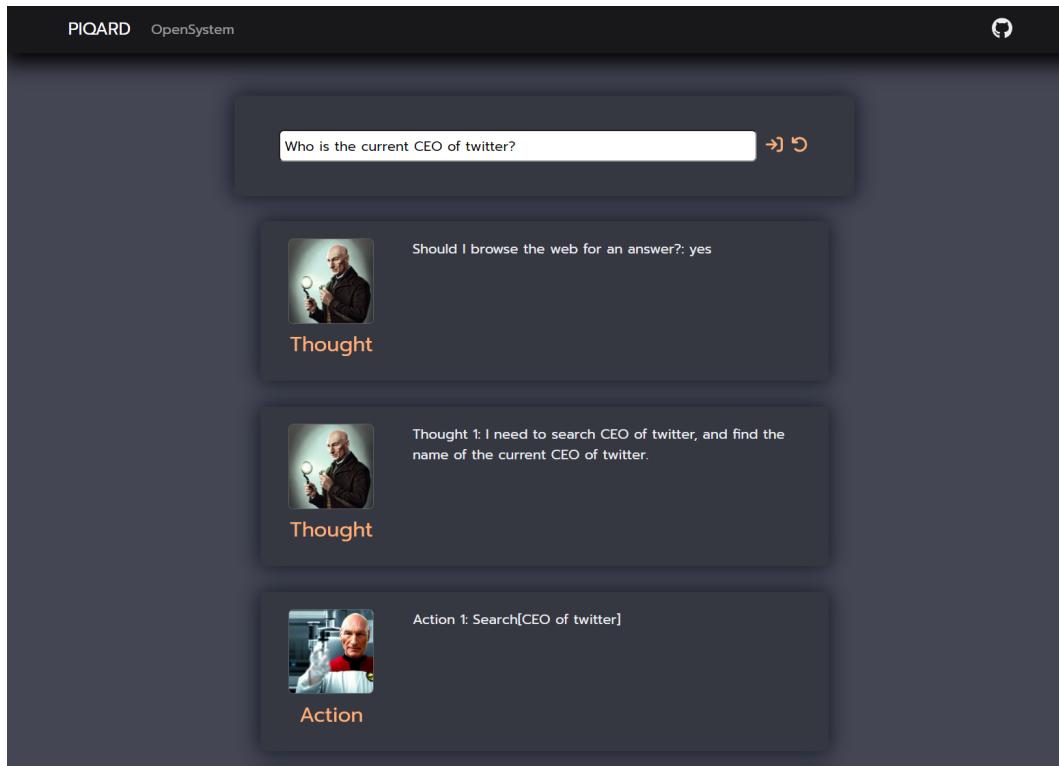
```

1 basic_prompt_template = """Answer the question.
2 Q: {{question}}
3 A:
4 """
5 basic_prompt = PromptTemplate(template=basic_prompt_template)
6 cohere = CohereAPI(stop_token="\n")
7
8 cohere.query(basic_prompt.render(question="Who is the current CEO of twitter?"))

```

'Jack Dorsey is the current CEO of twitter.'

RYSUNEK 10.10: Odpowiedź wygenerowana przez proste zapytanie modelu języka.



RYSUNEK 10.11: Odpowiedź wygenerowana przez system (część 1/2).



Observation

Observation: Elon Reeve Musk (EE-Ion; born June 28, 1971) is a business magnate and investor. He is the founder, CEO and chief engineer of SpaceX; angel investor, CEO and product architect of Tesla, Inc.; owner and CEO of Twitter, Inc.; founder of The Boring Company; co-founder of Neuralink and OpenAI; and president of the philanthropic Musk Foundation. With an estimated net worth of around \$139 billion as of December 23, 2022, primarily from his ownership stakes in Tesla and SpaceX, Musk is the second-wealthiest person in the world, according to both the Bloomberg Billionaires Index and Forbes's real-time billionaires list. Musk was born in Pretoria, South Africa, and briefly attended at the University of Pretoria before moving to Canada at age 18, acquiring citizenship through his Canadian-born mother. Two years later, he matriculated at Queen's University and transferred to the University of Pennsylvania, where he received bachelor's degrees in economics and physics. He moved to California in 1995 to attend Stanford University. After two days, he dropped out and with his brother Kimbal, co-founded the online city guide software company Zip2. In 1999, Zip2 was acquired by Compaq for \$307 million and Musk co-founded X.com, a direct bank. X.com merged with Confinity in 2000 to form PayPal, which eBay acquired for \$15 billion in 2002. With \$175.8 million, Musk founded SpaceX in 2002, a spaceflight services company. In 2004, he was an early investor in the electric vehicle manufacturer Tesla Motors, Inc.



Thought

Thought 2: Elon Musk is the current CEO of twitter.



Action

Action 2: Finish[Elon Musk]



Answer

Elon Musk

PIQARD Open System

Try again ⚡

RYSUNEK 10.12: Odpowiedź wygenerowana przez system (część 2/2).

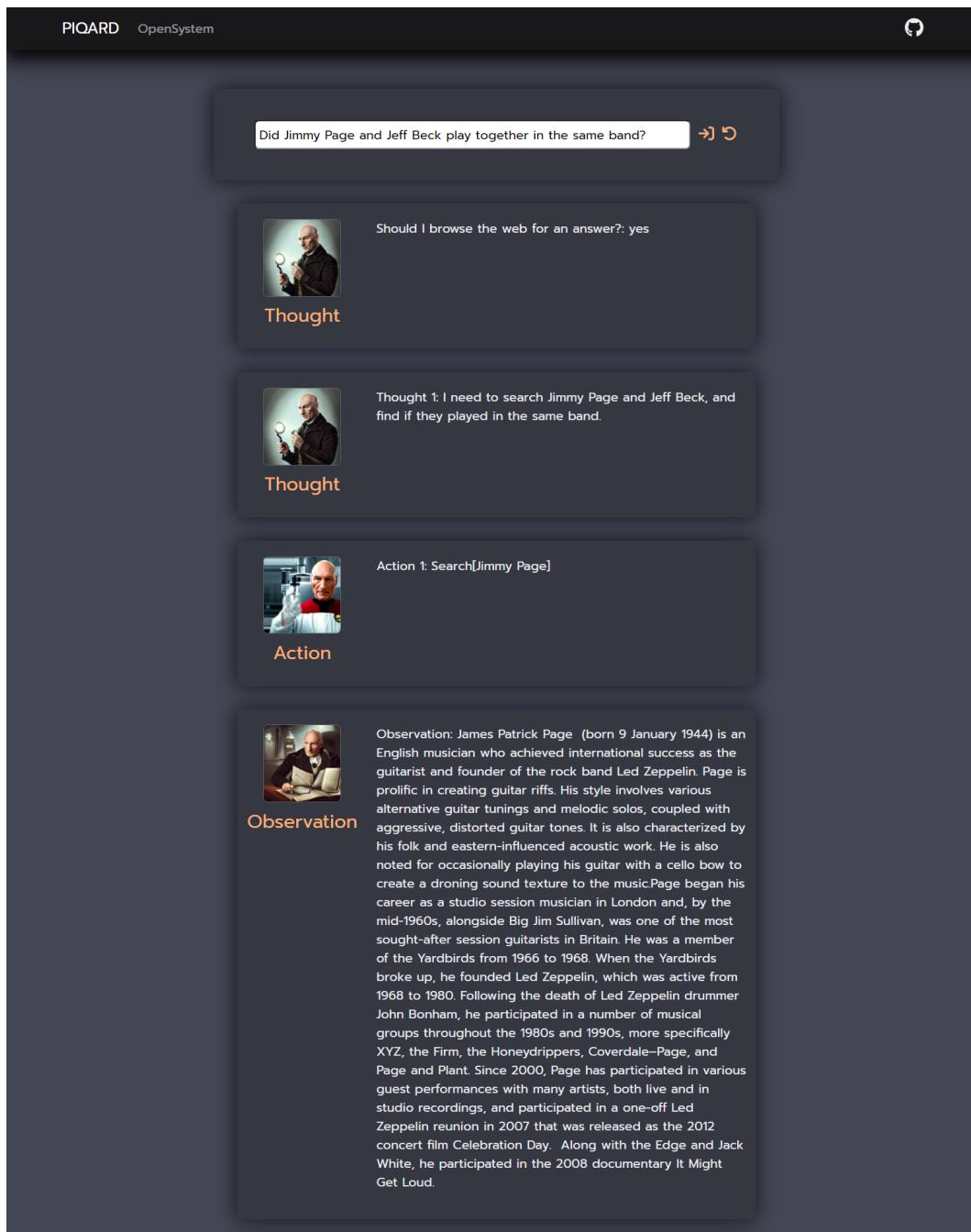
- **Pytanie: Did Jimmy Page and Jeff Beck play together in the same band?**

Zaproponowany system udzielił poprawną odpowiedź oraz podał źródło, z którego skorzystał, aby zaktualizować swoją wiedzę (rys. 10.14, 10.15). Natomiast odpowiedź wygenerowana przez proste odpytanie modelu języka jest niepoprawna (rys. 10.13). Obaj artyści grali przez pewien czas w tym samym zespole "The Yardbirds".

```
1 basic_prompt_template = """Answer the question.
2 Q: {{question}}
3 A:
4 """
5 basic_prompt = PromptTemplate(template=basic_prompt_template)
6 cohere = CohereAPI(stop_token="\n")
7
8 cohere.query(basic_prompt.render(question="Did Jimmy Page and Jeff Beck play in the same rock band?"))
```

'No, they were in different rock bands.'

RYSUNEK 10.13: Odpowiedź wygenerowana przez proste zapytanie modelu języka.



RYSUNEK 10.14: Odpowiedź wygenerowana przez system (część 1/2).

The diagram illustrates a process flow:

- Thought:** A small image of a man holding a magnifying glass. Next to it is the text: "Thought 2: Jeff Beck is a guitarist. I need to search if he played with Jimmy Page in the same band."
- Action:** A small image of a man in a lab coat. Next to it is the text: "Action 2: Search[Jeff Beck]".
- Observation:** A larger text block containing information about Geoffrey Arnold Beck (1944-2023), his career with The Yardbirds and the Jeff Beck Group, and his musical styles. It also mentions his induction into the Rock and Roll Hall of Fame.
- Thought:** A small image of a man holding a magnifying glass. Next to it is the text: "Thought 3: Jimmy Page and Jeff Beck were members of the Yardbirds. So they played in the same band."
- Action:** A small image of a man in a lab coat. Next to it is the text: "Action 3: Finish[yes]".
- Answer:** A box containing the word "yes".

PIQARD Open System
Try again ↶

RYSUNEK 10.15: Odpowiedź wygenerowana przez system (część 2/2).

- **Pytanie: Did Aristotle use a laptop?**

Zaproponowany system udzielił poprawną odpowiedź oraz podał źródło, z którego skorzystał, aby zaktualizować swoją wiedzę (rys. 10.17, 10.18). Odpowiedź wygenerowana przez proste odpytanie modelu języka również jest poprawna (rys. 10.16), jednak należy mieć na uwadze, że użytkownik nie posiada jej żadnego potwierdzenia, czyli nie może wykluczyć halucynacji wiedzy.

```
1 basic_prompt_template = """Answer the question.
2 Q: {{question}}
3 A:
4 """
5 basic_prompt = PromptTemplate(template=basic_prompt_template)
6 cohere = CohereAPI(stop_token="\n")
7
8 cohere.query(basic_prompt.render(question="Did Aristotle use a laptop?"))
```

'No, he used a scroll.'

RYSUNEK 10.16: Odpowiedź wygenerowana przez proste zapytanie modelu języka.



RYSUNEK 10.17: Odpowiedź wygenerowana przez system (część 1/2).

 Thought 2: Laptop was first made in 1981. I need to search Aristotel next.

Thought

 Action 2: Search[Aristote]

Action

 Observation: Aristotle (; Greek: Ἀριστοτέλης Aristotélēs, pronounced [aristoté:lɛ:s]; 384–322 BC) was an Ancient Greek philosopher and polymath. His writings cover a broad range of subjects including physics, biology, zoology, metaphysics, logic, ethics, aesthetics, poetry, drama, music, rhetoric, psychology, linguistics, economics, politics, meteorology, geology, and government. As the founder of the Peripatetic school of philosophy in the Lyceum in Athens, he began the wider Aristotelian tradition that followed, which set the groundwork for the development of modern science. Little is known about Aristotle's life. He was born in the city of Stagira in Northern Greece during the Classical period. His father, Nicomachus, died when Aristotle was a child, and he was brought up by a guardian. At seventeen or eighteen years of age he joined Plato's Academy in Athens and remained there until the age of thirty-seven (c. 347 BC). Shortly after Plato died, Aristotle left Athens and, at the request of Philip II of Macedon, tutored his son Alexander the Great beginning in 343 BC. He established a library in the Lyceum which helped him to produce many of his hundreds of books on papyrus scrolls. Though Aristotle wrote many elegant treatises and dialogues for publication, only around a third of his original output has survived, none of it intended for publication. Aristotle provided a complex synthesis of the various philosophies existing prior to him.

 Thought 3: Laptop was made in 1981, and Aristotle died in 322 BC. So Aristotle didn't use a laptop.

Thought

 Action 3: Finish[no]

Action

 no

Answer

PIQARD Open System

Try again ☺

RYSUNEK 10.18: Odpowiedź wygenerowana przez system (część 2/2).

Rozdział 11

Podsumowanie

W momencie rozpoczęcia pracy nad naszą rozprawą dyplomową systemy odpowiadające na zadane pytanie, które wykorzystują modele języka naturalnego, i również biblioteki stworzone na potrzebę eksperymentowania z pojawiającymi się strategiami promptingu, nie cieszyły się jeszcze bardzo dużą popularnością. Wraz z rozwojem naszego projektu w ciągu ostatnich miesięcy zauważaliśmy jednak rosnące zainteresowanie powyższymi zagadnieniami wśród społeczności naukowej, co doprowadziło do powstania i udostępniania nowych narzędzi. Przykładami są, chociażby platforma *Perplexity AI*¹, która umożliwia wielodziedzinowe odpytanie systemu z wyszukiwaniem informacji lub biblioteka *LangChain*² pozwalająca tworzyć aplikacje bazujące na zaimplementowanych strategiach promptingu. Sądzimy, że nasza praca ma związek z założeniami obu projektów, lecz jednocześnie wyraźnie rozszerza je o przeprowadzone eksperymenty dotyczące wpływu wybranych strategii na generowane odpowiedzi.

Głównym celem naszej pracy było stworzenie systemu wykorzystującego neuronowe modele języka naturalnego, który udziela odpowiedzi na zadane pytania wzbogacone o wskazany kontekst w dokumentach. W naszej opinii zamierzony cel został osiągnięty, a wszystkie początkowe założenia i szczegółowe pod-cele uważamy za spełnione. Główne osiągnięcia można podsumować następująco:

1. Stworzyliśmy bibliotekę w języku Python, która umożliwia programistom łatwą budowę systemu odpowiadającego na pytania z zaimplementowanych przez nas komponentów. Udostępnia ona m.in. modele języka naturalnego BLOOM 176B oraz Cohere xlarge, metody wyszukiwania informacji BM25, FAISS i ANNOY, a także klasę PIQARD, która stanowi interfejs łączący ze sobą działanie poszczególnych obiektów. Dodatkowo przygotowaliśmy implementacje rozszerzonych strategii promptingu ReAct oraz proponowanej przez nas autorskiej strategii SelfAware, która polega na początkowym odpytaniu modelu języka odnośnie potrzeby uzyskania dodatkowego kontekstu dla pytania użytkownika. Przygotowaliśmy również testy jednostkowe umożliwiające utrzymanie oraz dalszy rozwój projektu.
2. Wspomniana powyżej biblioteka zapewniła nam możliwość propotypowania oraz testowania promptów poprzez tworzenie środowisk zbudowanych z zaimplementowanych komponentów. Dzięki modułowej strukturze możliwe jest intuicyjne dostosowywanie ich działania do danej strategii promptingu. Dodatkowo zaimplementowana klasa wczytująca prompt pozwala na łatwą podmianę jego zawartości zarówno z pliku, jak i podanego ciągu znaków, co przyspiesza przeprowadzanie badań. Udostępniliśmy również przykładowe prompty przygotowane przez nas dla różnych strategii i zbiorów danych.

¹<https://www.perplexity.ai/>

²<https://github.com/hwchase17/langchain>

3. Zaprojektowaliśmy także aplikację internetową w języku React.js, która udostępnia interfejs użytkownika pozwalający na interakcję z zaproponowaną konfiguracją systemu zbudowanego w oparciu o bibliotekę. Zapewnia ona możliwość zadania pytania oraz analizy otrzymanej odpowiedzi wraz ze szczegółami. Dzięki temu, z możliwości oferowanych przez stosowanie strategii promptingu mogą korzystać także mniej techniczni użytkownicy, co ma potencjał wpłygnięcia na popularyzację opisywanych rozwiązań.
4. Naszą pracę wzbogaciliśmy również o przeprowadzone eksperymenty na benchmarkach OpenBookQA oraz HotpotQA mające na celu przetestowanie wpływu wybranych strategii promptingu na generowanie odpowiedzi przez modele języka BLOOM 176B i Cohere xlarge, m.in. łańcucha wnioskowania oraz dynamicznego promptingu. Zaobserwowałyśmy pozytywny wpływ ich stosowania na osiągane wyniki, lecz jednocześnie zwróciłyśmy uwagę na różnice pomiędzy wartościami dokładnego dopasowania wynikające z użytych promptów, modeli języka oraz benchmarków. Jedną z ważniejszych obserwacji, jest potrzeba dobrania strategii do charakterystyki zadawanych pytań oraz wybranego modelu języka.

Całość naszej pracy, na którą składa się stworzona biblioteka w języku Python wraz z aplikacją internetową w języku React.js, zamieściłyśmy w formie open-source w repozytorium Github pod linkiem <https://github.com/plaskod/piqard>. W związku z limitem czasu ograniczyliśmy zaimplementowane strategie promptingu oraz metody wyszukiwania, jednak uważamy, że możliwe jest rozszerzenie obecnych komponentów także przez członków społeczności.

Poniżej przedstawiamy dodatkowe obserwacje oraz refleksje, które towarzyszyły nam w trakcie tworzenia pracy, a następnie proponujemy potencjalne dalsze kierunki rozwoju.

11.1 Forma przeprowadzonych eksperymentów

Przeprowadzenie eksperymentów okazało się wyzwaniem w związku z ograniczonym czasem oraz zasobami, dlatego wybrałyśmy jedynie dwa benchmarki tj. OpenBookQA i HotpotQA. Dodatkowo ograniczyliśmy liczbę pytań drugiego z nich, w związku z czym osiągane wyniki nie mogą być dokładnie porównane z wynikami innych rozwiązań. Sądzimy, że moglibyśmy powtórzyć badania na pełnym zbiorze, a także zastanawialiśmy się nad wyborem innych benchmarków, które być może pozwoliłyby na lepszą ocenę wykorzystania kontekstu ze strategią łańcucha wnioskowania jak np. LAMA [PRL⁺19] lub TruthfulQA [LHE21]. Chcielibyśmy także wspomnieć, iż nie zdążyliśmy przeprowadzić wystarczających testów dla rozszerzonych strategii ReAct oraz SelfAware.

Warto również dodać, że nie zakładaliśmy trenowania modeli języka, gdyż dla wybranych wariantów z dużą liczbą parametrów było to dla nas niewykonalne w związku z posiadanymi zasobami. Pomimo możliwości nie chcieliśmy także skupić się na mniejszych, dostępnych dla nas modelach, gdyż prawa skalowania sugerują, że wykorzystanie niektórych strategii, jak np. łańcuch wnioskowania, zachodzą dopiero dla większej liczby parametrów.

11.2 Wybrane modele języka

1. BLOOM – Wyniki uzyskane z przeprowadzonych eksperymentów wskazują na pozytywny wpływ zastosowania metod uczenia w kontekście oraz wyszukiwania informacji. Niemniej, oczekiwania dotyczące skuteczności strategii łańcuchów wnioskowania w modelu BLOOM, o liczbie parametrów porównywalnej do GPT-3, nie zostały w pełni spełnione. Przypuszczamy, że przyczyną tego jest wielojęzyczny charakter zbioru danych, na którym model został wytre-

nowany, co mogło negatywnie wpływać na jego zdolność do zastosowania strategii łańcuchów wnioskowania w języku angielskim.

2. Cohere xlarge – Problematyczne jest zinterpretowanie zachowań tego modelu, gdyż autorzy nie zapewniają szczegółowych informacji dotyczących procesu treningu oraz zbioru danych użytych do jego przeprowadzenia.

11.2.1 Ograniczenia związane z oknem kontekstowym modeli języka

Zauważaliśmy, że jednym z istotnych ograniczeń jest ograniczone okno kontekstowe, które jest wynikiem złożoności obliczeniowej mechanizmu samouwagi. To z kolei ma wpływ na długość opisu zadania oraz liczbę przykładów, które mogą być uwzględnione w zadaniu. W przypadku modelu BLOOM, który został zastosowany za pośrednictwem interfejsu API, to ograniczenie zostało dodatkowo pogłębiane przez restrykcję długości sekwencji wejściowej do 1024 tokenów, co miało na celu zmniejszenie rozmiaru danych przesyłanych przy każdym wywołaniu API. Ta sytuacja jest najbardziej dotkliwa w implementacji zaawansowanych strategii promptingu, które polegają na wzbudzaniu modelu do generowania pośrednich kroków poprzez udzielanie odpowiedzi (np. łańcuchy wnioskowania, SelfAware, self-ask, ReAct), gdzie liczba tokenów w każdym z przykładów jest większa w porównaniu z metodami takimi jak few-shot. W implementacji strategii ReAct, z powodu tych ograniczeń, musielibyśmy zrezygnować z funkcjonalności takiej jak *Lookup*, ponieważ liczba tokenów była wystarczająca tylko na kilka przykładów wykonywania trajektorii z akcją *Search*, a podawanie innych typów akcji znacząco pogarszało działanie strategii.

11.2.2 Ograniczenia związane z korzystaniem z API

Rzykiem związanym z badaniem modeli języka, gdy dostęp do nich jest możliwy tylko przez zewnętrzne API, jest brak kontroli nad ich przechowywaniem i korzystaniem z nich. W trakcie przeprowadzonych eksperymentów zauważaliśmy następujące kwestie:

1. Przeprowadzana diagnostyka modelu BLOOM na platformie Azure uniemożliwiła wykonanie eksperymentów przez okres serwisowania.
2. Restrykcyjne ograniczenie długości sekwencji wejściowej dla modelu BLOOM.
3. Blokowanie przez API Cohere przetwarzania zapytań niezgodnych z polityką i regulaminem organizacji, co przeszkodziło w realizacji eksperymentów na HotpotQA.
4. Brak możliwości uzyskania rozkładu prawdopodobieństwa jako wynik przetwarzania modelu języka uniemożliwił stosowanie innych metod próbkowania niż te dostarczone przez API.
5. Dokumentacja API na HuggingFace zawierała nieścisłości, a testowane parametry w API do modelu BLOOM nie zawsze działały zgodnie z oczekiwaniemi.

11.3 Strategie promptingu

W trakcie implementacji promptingu Self-ask i ReAct zauważaliśmy trudności związane z przenoszeniem tych strategii na inne zbiorы danych. Wynika to z tego, że te metody często są dostosowywane do specyficznego zbioru danych, na którym zostały opracowane. Aby zminimalizować to ograniczenie, zastosowaliśmy kompozycję obu strategii w naszym systemie OpenSystem, co pozwoliło na osiągnięcie lepszych rezultatów na pytaniach spoza dziedziny ich pierwotnego przeznaczenia i pokazało potencjał modeli językowych do realizacji złożonych strategii promptingu.

Należy również wziąć pod uwagę kwestię annotacji danych. Zauważaliśmy istotne różnice w wynikach osiąganych przez strategię łańcuchów wnioskowania, w zależności od osoby odpowiedzialnej za uzupełnienie przykładów w promptcie. Nasze obserwacje są spójne z literaturą o łańcuchach wnioskowania [CND⁺22]. Jest to przykład wysokiej wrażliwości modeli języka na sformułowania i słowa zawarte w promptcie, co ilustruje trudności związane z implementacją skutecznych promptów wzbudzających oczekiwane działanie.

11.4 Wyszukiwanie informacji

W naszej pracy korzystaliśmy z kilku metod wyszukiwania informacji. Miało to zapewnić możliwie najlepszy dobór dokumentów traktowanych jako kontekst, które umożliwiały udzielenie prawnej odpowiedzi na postawione pytanie. W trakcie przeprowadzanych eksperymentów na benchmarkach OpenBookQA oraz HotpotQA zaobserwowaliśmy, że wyszukiwanie semantyczne okazało się zwracać bardziej dopasowane dokumenty, niż wyszukiwanie leksykalne. Jednak dodatkowo zauważaliśmy również różnice w czasach uzyskania dokumentów dla dużych zbiorów danych takich jak HotpotQA. Najbardziej wydajny okazał się ANNOY i wynika to ze sposobu jego implementacji. Jest on napisany w języku C++ oraz korzysta z mechanizmu zwanego MMAP, przez co eliminuje potrzebę korzystania z dodatkowych wywołań systemowych typu odczyt lub zapis.

Warto zaznaczyć, że wybór zbioru danych, który będzie przeszukiwany, ma znaczenie dla jakości udzielonej odpowiedzi przez system i różne bazy dokumentów mogą być bardziej odpowiednie dla postrzególnych pytań. Wybranie nieodpowiedniego zbioru dokumentów może wiązać się z tym, że dana informacja nie zostanie odnaleziona, lub – co gorsza – uzyskamy fałszywą informację na temat pytania, jeśli źródło pochodzenia danych jest nierzetelne.

11.5 Dalsze kierunki rozwoju

W ramach przyszłych badań moglibyśmy przetestować nasz system na większej liczbie benchmarków, co pozwoliłoby na lepsze zobrazowanie jego zdolności w zakresie udzielania odpowiedzi na pytania. Dodatkowo chcielibyśmy rozszerzyć zaimplementowane strategie promptingu:

1. Rozszerzyć prompting dynamiczny o inne sposoby dobierania przykładów.
2. Wprowadzić nowe rodzaje akcji w strategii ReAct

W dalszej perspektywie uważamy za istotne przetestowanie strategii *self-consistency* [WWS⁺22a] w celu generowania różnorodnych łańcuchów wnioskowania przez niezachlanne metody próbkowania modelu języka. Takie podejście może również być stosowane do automatycznego konstruowania zbioru przykładów uczących, na których model generuje łańcuch wnioskowania prowadzący do prawidłowej odpowiedzi na zbiorze treningowym [HGH⁺22], celem późniejszego dostrojenia modelu języka. Sądzimy również, że wpływ permutacji przykładów w promptcie jest istotną kwestią dla strategii promptingów. Uważamy, że przeprowadzone przez nas eksperymenty nie wyczerpały w pełni tego zagadnienia, dlatego moglibyśmy je rozszerzyć.

Myślimy także, iż ważnym elementem naszej pracy jest część odpowiadająca za wyszukiwanie informacji. Od jakości jej działania zależy informatywność zwracanych dokumentów, co z kolei wpływa na wykorzystane strategie promptingu, w których podawany jest dodatkowy kontekst do pytania. Być może warto byłoby przetestować dostrajane oraz wybór modeli odpowiadających za wektoryzację zbiorów danych.

Jesteśmy również otwarci na współpracę, w związku z czym udostępniliśmy nasz projekt w formie open-source na platformie Github.

Literatura

- [AHS85] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.
- [Ala18] Jay Alammar. The Illustrated Transformer [Blog post].
<https://jalammar.github.io/illustrated-transformer/>, 2018.
- [ASA⁺22] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models, 2022.
- [BBC⁺22] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, et al. Rt-1: Robotics transformer for real-world control at scale. <https://arxiv.org/abs/2212.06817>, 2022.
- [Ber15] Erik Bernhardsson. Nearest neighbors and vector models – part 2 – algorithms and data structures. <https://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html>, 2015.
- [Ber22] Erik Bernhardsson. annoy. <https://github.com/spotify/annoy>, 2022.
- [BMR⁺20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.
- [BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, mar 2003.
- [CFWB17] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. <https://arxiv.org/abs/1704.00051>, 2017.
- [Cha07] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *Int. J. Math. Model. Meth. Appl. Sci.*, 1, 01 2007.
- [CND⁺22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, et al. PaLM: Scaling language modeling with pathways. <https://arxiv.org/abs/2204.02311>, 2022.
- [CSL⁺22] Stephanie C. Y. Chan, Adam Santoro, Andrew K. Lampinen, Jane X. Wang, et al. Data distributional properties drive emergent in-context learning in transformers, 2022.
- [DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep Bidirectional Transformers for Language Understanding.
<https://arxiv.org/abs/1810.04805>, 2018.
- [DSD⁺22] Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models secretly perform gradient descent as meta-optimizers, 2022.
- [EL20] Avia Efrat and Omer Levy. The turking test: Can language models understand instructions? *CoRR*, abs/2010.11982, 2020.
- [FG17] Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. In *Proceedings of the Workshop on Stylistic Variation*, pages 94–104, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

- [FLD18] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia, July 2018. Association for Computational Linguistics.
- [GMT⁺22] Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, et al. Improving alignment of dialogue agents via targeted human judgements, 2022.
- [Gro22] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*, 2022.
- [HBD⁺20] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.
- [HBF⁺18] Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649, Melbourne, Australia, July 2018. Association for Computational Linguistics.
- [HGH⁺22] Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. <https://arxiv.org/abs/2210.11610>, 2022.
- [HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [JDJ17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017.
- [JDS11] Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.
- [Jia19] Yuxing Jia. Attention mechanism in machine translation. *Journal of Physics: Conference Series*, 1314(1):012186, oct 2019.
- [KEKD22] Tomasz Korbak, Hady Elsahar, Germán Kruszewski, and Marc Dymetman. On reinforcement learning and distribution matching for fine-tuning language models with no catastrophic forgetting, 2022.
- [KMH⁺20] Jared Kaplan, Sam McCandlish, Tom Henighan, Benjamin Chess, et al. Scaling laws for neural language models, 2020.
- [LARC21] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.
- [LBM⁺21a] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *CoRR*, abs/2104.08786, 2021.
- [LBM⁺21b] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity, 2021.
- [LHE21] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2021.
- [MCKS18] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Conference on Empirical Methods in Natural Language Processing*, 2018.

- [MH17] Leland McInnes and John Healy. Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, nov 2017.
- [MHM18] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018.
- [MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.
- [PLP⁺20] Fabio Petroni, Patrick Lewis, Aleksandra Piktus, Tim Rocktäschel, et al. How context affects language models' factual predictions, 2020.
- [PRL⁺19] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. Language models as knowledge bases?, 2019.
- [PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.
- [PZM⁺22] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models, 2022.
- [RBL⁺21] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.
- [RG19] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.
- [RRS20] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? <https://arxiv.org/abs/2002.08910>, 2020.
- [RWC⁺18] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2018.
- [RZ09] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.
- [SB88] Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):323–328, 1988.
- [SFA⁺22] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, et al. Bloom: A 176b-parameter open-access multilingual language model, 2022.
- [SJP22] SJP. Słownik języka polskiego, 2022.
- [SLP⁺21] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.
- [SPC⁺21] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. <https://arxiv.org/abs/2104.07567>, 2021.
- [SZK⁺22] John Schulman, Barret Zoph, Christina Kim, Jacob Hilton, Jacob Menick, et al. ChatGPT: Optimizing Language Models for Dialogue. <https://openai.com/blog/chatgpt/qwe.sh/%2F../>, 2022.
- [TDFH⁺22] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, et al. LaMDA: Language Models for Dialog Applications. <https://arxiv.org/abs/2201.08239>, 2022.

- [TKC⁺22] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, et al. Galactica: A large language model for science. <https://arxiv.org/abs/2211.09085>, 2022.
- [vONR⁺22] Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, et al. Transformers learn in-context by gradient descent. <https://arxiv.org/abs/2212.07677>, 2022.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, et al. Attention is all you need. <https://arxiv.org/abs/1706.03762>, 2017.
- [Wan21] Ben Wang. Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX. <https://github.com/kingoflolz/mesh-transformer-jax>, May 2021.
- [Wei66] Joseph Weizenbaum. ELIZA a Computer Program For the Study of Natural Language Communication Between Man And Machine.
<http://web.stanford.edu/class/cs124/p36-weizenbaum.pdf>, 1966.
- [Wik22a] Wikipedia contributors. Okapi bm25 — Wikipedia, the free encyclopedia.
https://en.wikipedia.org/w/index.php?title=Okapi_BM25&oldid=1112011249, 2022.
[Online; accessed 30-December-2022].
- [Wik22b] Wikipedia contributors. Wikipedia — Wikipedia, the free encyclopedia.
<https://en.wikipedia.org/w/index.php?title=Wikipedia&oldid=1130235694>, 2022.
[Online; accessed 30-December-2022].
- [WWS⁺22a] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, et al. Self-consistency improves chain of thought reasoning in language models. <https://arxiv.org/abs/2203.11171>, 2022.
- [WWS⁺22b] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, et al. Chain of thought prompting elicits reasoning in large language models, 2022.
- [XRLM21] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. <https://arxiv.org/abs/2111.02080>, 2021.
- [YQZ⁺18] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.
- [YXL⁺19a] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, et al. End-to-end open-domain question answering with. In *Proceedings of the 2019 Conference of the North*. Association for Computational Linguistics, 2019.
- [YXL⁺19b] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, et al. End-to-end open-domain question answering with. In *Proceedings of the 2019 Conference of the North*. Association for Computational Linguistics, 2019.
- [YZY⁺22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, et al. React: Synergizing reasoning and acting in language models. <https://arxiv.org/abs/2210.03629>, 2022.
- [ZLW⁺21] Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. Retrieving and reading: A comprehensive survey on open-domain question answering. <https://arxiv.org/abs/2101.00774>, 2021.
- [ZWF⁺21] Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. <https://arxiv.org/abs/2102.09690>, 2021.

Dodatek A

Prompty wykorzystane w pracy

TABELA A.1: Bazowy prompt 0-shot

Answer the question.

Question: {{insert question}}

Answer:

TABELA A.2: Prompt 0-shot z konkatenacją kontekstów w języku jinja2

Answer the question given the context.

Question: {{insert question}}

Context: {% if context %}{{"join(context)}}{% endif %}

Answer:

TABELA A.3: Prompt generowany dynamicznie w języku jinja2 dla dowolnej liczby przykładów i faktów

Answer the question based on the facts.

{% for example in prompt.examples %}

Question: {{example['text']}}}

Facts: {% if example['context'] %}{% for fact in example['context'] %}

- {{fact}}{% endfor %}{% endif %}

Answer: {{example['answer']}}}

{% endfor %}

Question: {{insert question}}

Facts: {% if context %}{% for fact in context %}

- {{fact}}{% endfor %}{% endif %}

Answer:

TABELA A.4: Prompt w stylu systemu dialogowego

Assistant is a large language model.

Assistant is designed to be able to assist with a wide range of tasks, from answering simple questions to providing in-depth explanations and discussions on a wide range of topics.

As a language model, Assistant is able to generate human-like text based on the input it receives, allowing it to engage in natural-sounding conversations and provide responses that are coherent and relevant to the topic at hand.

Assistant is constantly learning and improving, and its capabilities are constantly evolving. It is able to process and understand large amounts of text, and can use this knowledge to provide accurate and informative responses to a wide range of questions.

Additionally, Assistant is able to generate its own text based on the input it receives, allowing it to engage in discussions and provide explanations and descriptions on a wide range of topics.

Overall, Assistant is a powerful tool that can help with a wide range of tasks and provide valuable insights and information on a wide range of topics.

Whether you need help with a specific question or just want to have a conversation about a particular topic, Assistant is here to assist.

Human: {{insert question}}

Assistant:

TABELA A.5: Prompt w stylu systemu dialogowego z wprowadzeniem kontekstu

Assistant is a large language model.

Assistant is designed to be able to assist with a wide range of tasks, from answering simple questions to providing in-depth explanations and discussions on a wide range of topics.

As a language model, Assistant is able to generate human-like text based on the input it receives, allowing it to engage in natural-sounding conversations and provide responses that are coherent and relevant to the topic at hand.

Assistant is constantly learning and improving, and its capabilities are constantly evolving. It is able to process and understand large amounts of text, and can use this knowledge to provide accurate and informative responses to a wide range of questions.

Additionally, Assistant is able to generate its own text based on the input it receives, allowing it to engage in discussions and provide explanations and descriptions on a wide range of topics.

Assistant may receive additional context from the user indicated by "Context:". It is using the received context to provide a more accurate and helpful response.

Overall, Assistant is a powerful tool that can help with a wide range of tasks and provide valuable insights and information on a wide range of topics.

Whether you need help with a specific question or just want to have a conversation about a particular topic, Assistant is here to assist.

Human: {{insert question}} **Context:** {% if context %}{{context}}{% endif %}

Assistant:

TABELA A.6: Prompt self-aware do podejmowania decyzji, czy pytanie wymaga wykorzystania źródeł zewnętrznych

For each input, determine if the answerer will need to access the internet to respond. Statements or question of opinion do not require the internet, factual question or commands might. In general, simple factual questions whose answers do not change over time can be answered without the internet access. Questions that refer to current events, news, sports, websites that change frequently, etc. will require the internet access. Questions that refer highly specific events that the answer may not have memorized also require the internet access. If they do not need to access the internet, the answerer will respond with "No". If they do, rephrase the question in the form of search command.

Input: Hello, how are you?

Output: No

Input: What is the capital of France?

Output: No

Input: Who won the 2022 World Cup?

Output: Find the 2022 Would Cup winner

Input: What's your name?

Output: No

Input: What is the current temperature in Paris?

Output: Find the current temerpature in Paris

Input: What is machine learning?

Output: No

Input: {{insert question}}

Output:

TABELA A.7: Prompt ReAct [YZY⁺²²]

Question: What profession does Nicholas Ray and Elia Kazan have in common?

Thought 1: I need to search Nicholas Ray and Elia Kazan, find their professions, then find the profession they have in common.

Action 1: *Search[Nicholas Ray]*

Observation: Nicholas Ray (born Raymond Nicholas Kienzle Jr., August 7, 1911 - June 16, 1979) was an American film director, screenwriter, and actor best known for the 1955 film *Rebel Without a Cause*.

Thought 2: Professions of Nicholas Ray are director, screenwriter, and actor. I need to search Elia Kazan next and find his professions.

Action 2: *Search[Elia Kazan]*

Observation: Elia Kazan was an American film and theatre director, producer, screenwriter and actor.

Thought 3: Professions of Elia Kazan are director, producer, screenwriter, and actor. So profession Nicholas Ray and Elia Kazan have in common is director, screenwriter, and actor.

Action 3: *Finish[director, screenwriter, actor]*

Question: Which magazine was started first Arthur's Magazine or First for Women?

Thought 1: I need to search Arthur's Magazine and First for Women, and find which was started first.

Action 1: *Search[Arthur's Magazine]*

Observation: Arthur's Magazine (1844-1846) was an American literary periodical published in Philadelphia in the 19th century.

Thought 2: Arthur's Magazine was started in 1844. I need to search First for Women next.

Action 2: *Search[First for Women]*

Observation: First for Women is a woman's magazine published by Bauer Media Group in the USA.^[1] The magazine was started in 1989.

Thought 3: First for Women was started in 1989. 1844 < 1989, so Arthur's Magazine was started first.

Action 3: *Finish[Arthur's Magazine]*

Question: What is the longest bridge in the world?

Thought 1: I need to search the name of the longest bridge in the world.

Action 1: *Search[the longest bridge in the world]*

Observation: The Danyang-Kunshan Grand Bridge is officially the longest bridge in the world. It stretches out for no less than 102.4 miles (164.8 kilometres) and forms part of one of China's most important railway connections: the high-speed train between Beijing and Shanghai.

Action 2: *Finish[The Danyang-Kunshan Grand Bridge]*

Question: Were Pavel Urysohn and Leonid Levin known for the same type of work?

Thought 1: I need to search Pavel Urysohn and Leonid Levin, find their types of work, then find if they are the same.

Action 1: *Search[Pavel Urysohn]*

Observation: Pavel Samuilovich Urysohn (February 3, 1898 - August 17, 1924) was a Soviet mathematician who is best known for his contributions in dimension theory.

Thought 2: Pavel Urysohn is a mathematician. I need to search Leonid Levin next and find its type of work.

Action 2: *Search[Leonid Levin]*

Observation: Leonid Anatolievich Levin is a Soviet-American mathematician and computer scientist.

Thought 3: Leonid Levin is a mathematician and computer scientist. So Pavel Urysohn and Leonid Levin have the same type of work.

Action 3: *Finish[yes]*

Question: {{insert question}}

TABELA A.8: OpenBookQA, prompt 0-shot z dowolną liczbą faktów

Answer the question based on the facts.

Question: {{insert question}}

Possible answers: {{possible answers}}

Facts: {% if context %}{% for fact in context %}

- {{fact}}{% endfor %}{% endif %}

Answer:

TABELA A.9: OpenBookQA, prompt 5-shot z podanym faktem

Answer the question based on the facts.

Question: The sun is responsible for

Possible answers: A. puppies learning new tricks, B. children growing up and getting old, C. flowers wilting in a vase, D. plants sprouting, blooming and wilting

Facts:

- the sun is the source of energy for physical cycles on Earth

Answer: D. plants sprouting, blooming and wilting

Question: with which could you tell the exact size of an object?

Possible answers: A. a plain stick with irregular shape, B. a plastic tape with graduated markings, C. a thermometer with mercury in it, D. a metal cooking spoon

Facts:

- a tape measure is used to measure length

Answer: B. a plastic tape with graduated markings

Question: When food is reduced in the stomach

Possible answers: A. the mind needs time to digest, B. take a second to digest what I said, C. nutrients are being deconstructed, D. reader's digest is a body of works

Facts:

- digestion is when stomach acid breaks down food

Answer: C. nutrients are being deconstructed

Question: Stars are

Possible answers: A. warm lights that float, B. made out of nitrate, C. great balls of gas burning billions of miles away, D. lights in the sky

Facts:

- a star is made of gases

Answer: C. great balls of gas burning billions of miles away

Question: Poison causes harm to which of the following?

Possible answers: A. a tree, B. a robot, C. a house, D. a car

Facts:

- poison causes harm to living things

Answer: A. a tree

Question: {{insert question}}

Possible answers: {{possible answers}}

Facts: {% if context %}{% for fact in context %}

- {{fact}}{% endfor %}{% endif %}

Answer:

TABELA A.10: OpenBookQA, prompt 5-shot z trzema faktami

Answer the question based on the facts.

Question: The sun is responsible for

Possible answers: A. puppies learning new tricks, B. children growing up and getting old, C. flowers wilting in a vase, D. plants sprouting, blooming and wilting

Facts:

- the sun is the source of energy for physical cycles on Earth
- the sun is the source of energy for life on Earth
- the sun is a source of heat called sunlight

Answer: D. plants sprouting, blooming and wilting

Question: with which could you tell the exact size of an object?

Possible answers: A. a plain stick with irregular shape, B. a plastic tape with graduated markings, C. a thermometer with mercury in it, D. a metal cooking spoon

Facts:

- a tape measure is used to measure length
- a ruler is used for measuring the length of an object
- as the size of an object appears larger , that object will be observed better

Answer: B. a plastic tape with graduated markings

Question: When food is reduced in the stomach

Possible answers: A. the mind needs time to digest, B. take a second to digest what I said, C. nutrients are being deconstructed, D. reader's digest is a body of works

Facts:

- digestion is when stomach acid breaks down food
- the breaking down of food into simple substances occurs in the digestive system
- the digestive system breaks food into nutrients for the body

Answer: C. nutrients are being deconstructed

Question: Stars are

Possible answers: A. warm lights that float, B. made out of nitrate, C. great balls of gas burning billions of miles away, D. lights in the sky

Facts:

- a star is made of gases
- the stars in the night sky are very far away from the Earth
- nuclear reactions in stars causes stars to produce light

Answer: C. great balls of gas burning billions of miles away

Question: Poison causes harm to which of the following?

Possible answers: A. a tree, B. a robot, C. a house, D. a car

Facts:

- poison causes harm to living things
- harming something has a negative effect on that something
- polluting means something poisonous is added to an environment

Answer: A. a tree

Question: {{insert question}}

Possible answers: {{possible answers}}

Facts: {% if context %}{% for fact in context %}

- {{fact}}{% endfor %}{% endif %}

Answer:

TABELA A.11: OpenBookQA, prompt 5-shot z permutacją kolejności przykładów

Answer the question based on the facts.

Question: Stars are

Possible answers: A. warm lights that float, B. made out of nitrate, C. great balls of gas burning billions of miles away, D. lights in the sky

Facts:

- a star is made of gases

Answer: C. great balls of gas burning billions of miles away

Question: Poison causes harm to which of the following?

Possible answers: A. a tree, B. a robot, C. a house, D. a car

Facts:

- poison causes harm to living things

Answer: A. a tree

Question: The sun is responsible for

Possible answers: A. puppies learning new tricks, B. children growing up and getting old, C. flowers wilting in a vase, D. plants sprouting, blooming and wilting

Facts:

- the sun is the source of energy for physical cycles on Earth
- the sun is the source of energy for life on Earth
- the sun is a source of heat called sunlight

Answer: D. plants sprouting, blooming and wilting

Question: with which could you tell the exact size of an object?

Possible answers: A. a plain stick with irregular shape, B. a plastic tape with graduated markings, C. a thermometer with mercury in it, D. a metal cooking spoon

Facts:

- a tape measure is used to measure length
- a ruler is used for measuring the length of an object
- as the size of an object appears larger , that object will be observed better

Answer: B. a plastic tape with graduated markings

Question: When food is reduced in the stomach

Possible answers: A. the mind needs time to digest, B. take a second to digest what I said, C. nutrients are being deconstructed, D. reader's digest is a body of works

Facts:

- digestion is when stomach acid breaks down food

Answer: C. nutrients are being deconstructed

Question: {{insert question}}

Possible answers: {{possible answers}}

Facts: {% if context %}{% for fact in context %}

- {{fact}}{% endfor %}{% endif %}

Answer:

TABELA A.12: OpenBookQA, prompt 5-shot z permutacją kolejności przykładów

Answer the question based on the facts.

Question: The sun is responsible for

Possible answers: A. puppies learning new tricks, B. children growing up and getting old, C. flowers wilting in a vase, D. plants sprouting, blooming and wilting

Facts:

- the sun is the source of energy for physical cycles on Earth

Answer: D. plants sprouting, blooming and wilting

Question: with which could you tell the exact size of an object?

Possible answers: A. a plain stick with irregular shape, B. a plastic tape with graduated markings, C. a thermometer with mercury in it, D. a metal cooking spoon

Facts:

- a tape measure is used to measure length

Answer: B. a plastic tape with graduated markings

Question: When food is reduced in the stomach

Possible answers: A. the mind needs time to digest, B. take a second to digest what I said, C. nutrients are being deconstructed, D. reader's digest is a body of works

Facts:

- digestion is when stomach acid breaks down food

Answer: C. nutrients are being deconstructed

Question: Stars are

Possible answers: A. warm lights that float, B. made out of nitrate, C. great balls of gas burning billions of miles away, D. lights in the sky

Facts:

- a star is made of gases

Answer: C. great balls of gas burning billions of miles away

Question: Poison causes harm to which of the following?

Possible answers: A. a tree, B. a robot, C. a house, D. a car

Facts:

- poison causes harm to living things

Answer: A. a tree

Question: {{question}}

Possible answers: {{possible_answers}}

Facts: {% if context %}{% for fact in context %} - {{fact}}{% endfor %}{% endif %}

Answer:

TABELA A.13: OpenBookQA, prompt 5-shot z łańcuchem wnioskowania

Answer the question

Question: The sun is responsible for

Possible answers: A. puppies learning new tricks, B. children growing up and getting old, C. flowers wilting in a vase, D. plants sprouting, blooming and wilting

Let's think step by step:

- puppies can learn new tricks even in the dark or inside a house
- children growing up and getting old is a natural cycle of life
- flowers wilting in a vase could result from not supplying enough water
- sunlight is essential for plants for photosynthesis which make them grow, sprout, bloom and wilt when there's not enough of it

So the final answer is: D. plants sprouting, blooming and wilting

|||

Question: A magnet will stick to

Possible answers: A. a belt buckle, B. a wooden table, C. a plastic cup, D. a paper plate

Let's think step by step:

- a magnet will attract magnetic metals through magnetism
- a belt buckle is usually made out of metal such as nickel, zinc or copper alloys which are magnetic
- wood, plastic or paper are materials that do not contain metal and thus are not magnetic

So the final answer is: A. a belt buckle

|||

Question: Which would be a logical hypothesis after viewing a white substance on the floor, with a yellow carton on the counter?

Possible answers: A. milk was spilled, B. white juice was spilled, C. bleach was spilled, D. was an illusion

Let's think step by step:

- hypothesis means scientific guess about the cause and effect of an event
- the yellow carton on the counter was most likely found in the kitchen
- milk is a white liquid which is usually stored in carton boxes
- white juice is not very specific
- bleach is a white liquid but it's unlikely to be found on a counter
- let's not consider the possibility of being mistaken about seeing the white substance on the floor and it being just an illusion

So the final answer is: A. milk was spilled

|||

Question: A heavier object

Possible answers: A. requires less force to move, B. requires minimal effort to move, C. requires more muscle power to shift, D. requires a light touch to move

Let's think step by step:

- classical mechanics deals with the motion of bodies under the influence of forces or with the equilibrium of bodies when all forces are balanced
- Newton's second law is $F=ma$, where F is the force applied to the body, m is mass and a is acceleration
- consider a body with mass m₁ and a heavier body of mass m₂ = m₁+1, the force required to move the heavier body needs to be greater according to the Newton's second law
- force, effort, muscle power, touch in possible answers are the same words for making the object move

So the final answer is: C. requires more muscle power to shift

|||

Question: Which is most harmful to a human?

Possible answers: A. dynamite going off, B. reading email, C. balloons popping, D. opening a soda

Let's think step by step:

- a person may get killed if they're too close of a dynamite explosion, the shockwave can also damage hearing
- reading an email is not dangerous, at worst the email contains a computer virus which does not directly impact human's health
- balloons popping can be very irritating, but do not lead to hearing loss
- it's very unlikely to be harmed while opening a soda, at worst you cut your finger on a piece of aluminium

So the final answer is: A. dynamite going off

|||

Question: {{question}}

Possible answers: {{possible answers}}

Let's think step by step:

TABELA A.14: HotpotQA, prompt 5-shot z dwoma podanymi faktami

Given the following question and contexts provide an answer:

Question: What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?

Context: The Colorado orogeny, or Colorado orogen, was an orogeny in Colorado and surrounding areas which was a part of the development of the ancestral Rockies. The Colorado orogeny, formerly called the Colorado province, is a >500-km-wide belt of oceanic arc rock (1.78–1.65 Ga) that extends southward into New Mexico and composes a major part of the Proterozoic provinces of southwestern United States. This transcontinental collisional event occurred during the Paleoproterozoic (Statherian Period). The Wyoming sector of the Colorado orogeny was formerly called the Medicine Bow orogeny. The High Plains are a subregion of the Great Plains mostly in the Western United States, but also partly in the Midwest states of Nebraska, Kansas, and South Dakota, generally encompassing the western part of the Great Plains before the region reaches the Rocky Mountains. The High Plains are located in southeastern Wyoming, southwestern South Dakota, western Nebraska, eastern Colorado, western Kansas, eastern New Mexico, western Oklahoma, and south of the Texas Panhandle. The southern region of the Western High Plains ecology region contains the geological formation known as Llano Estacado which can be seen from a short distance or from on satellite maps. 7,000 ft, so the answer is 1,800 to 7,000 ft.

Answer: 1,800 to 7,000 ft

Question: Musician and satirist Allie Goertz wrote a song about the "The Simpsons" character Milhouse, who Matt Groening named after who?

Context: Allison Beth Ållie Goertz (born March 2, 1991) is an American musician. Goertz is known for her satirical songs based on various pop culture topics. Her videos are posted on YouTube under the name of Cossbysweater. Subjects of her songs have included the film *The Room*; the character Milhouse from the television show *The Simpsons*; and the game *Dungeons & Dragons*. Her style has been compared to that of Bo Burnham. Milhouse Mussolini van Houten is a fictional character featured in the animated television series *The Simpsons*; voiced by Pamela Hayden, and created by Matt Groening who named the character after President Richard Nixon's middle name. Later in the series, it is revealed that Milhouse's middle name is Mussolini.

Answer: Richard Nixon

Question: Which documentary is about Finnish rock groups, Adam Clayton Powell or The Saimaa Gesture?

Context: Adam Clayton Powell is a 1989 American documentary film directed by Richard Kilberg about the civil rights leader. It was nominated for an Academy Award for Best Documentary Feature. The Saimaa Gesture (Finnish: *Saimaa-ilmiöfö*) is a 1981 film by Finnish directors Aki and Mika Kaurismäki. It is a documentary of three Finnish rock groups aboard the steamboat SS Heinöe4vesi on their tour around Lake Saimaa.

Answer: The Saimaa Gesture

Question: What profession does Nicholas Ray and Elia Kazan have in common?

Context: Nicholas Ray (born Raymond Nicholas Kienzle Jr., August 7, 1911 – June 16, 1979) was an American film director best known for the movie *Rebel Without a Cause*. Elia Kazan (born Elias Kazantzoglou September 7, 1909 – September 28, 2003) was a Greek-American director, producer, writer and actor, described by *The New York Times* as one of the most honored and influential directors in Broadway and Hollywood history.

Answer: director, screenwriter, actor

Question: Which magazine was started first Arthur's Magazine or First for Women?

Context: Arthur's Magazine (1844–1846) was an American literary periodical published in Philadelphia in the 19th century. Edited by T.S. Arthur, it featured work by Edgar A. Poe, J.H. Ingraham, Sarah Josepha Hale, Thomas G. Spear, and others. In May 1846 it was merged into *Godey's Lady's Book*. First for Women is a woman's magazine published by Bauer Media Group in the USA. The magazine was started in 1989. It is based in Englewood Cliffs, New Jersey. In 2011 the circulation of the magazine was 1,310,696 copies.

Answer: Arthur's Magazine

Question: {{question}}

Context: {% if context %}{{".join(context)}}{% endif %}

Answer:

TABELA A.15: HotpotQA, prompt 5-shot z łańcuchem wnioskowania

Given the following question, write the step-by-step thought process and provide an answer:

Question: What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?

Thought: Let's think step by step. The eastern sector of Colorado orogeny extends into the High Plains. High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft.

Answer: 1,800 to 7,000 ft

|||

Question: Musician and satirist Allie Goertz wrote a song about the "The Simpsons" character Milhouse, who Matt Groening named after who?

Thought: Let's think step by step. Milhouse was named after U.S. president Richard Nixon, so the answer is Richard Nixon.

Answer: Richard Nixon

|||

Question: Which documentary is about Finnish rock groups, Adam Clayton Powell or The Saimaa Gesture?

Thought: Let's think step by step. Adam Clayton Powell (film) is a documentary about an African-American politician, not Finnish rock groups. So the documentary about Finnish rock groups must instead be The Saimaa Gesture.

Answer: The Saimaa Gesture

|||

Question: What profession does Nicholas Ray and Elia Kazan have in common?

Thought: Let's think step by step. Professions of Nicholas Ray are director, screenwriter, and actor. Professions of Elia Kazan are director, producer, screenwriter, and actor. So profession Nicholas Ray and Elia Kazan have in common is director, screenwriter, and actor.

Answer: director, screenwriter, actor

|||

Question: Which magazine was started first Arthur's Magazine or First for Women?

Thought: Let's think step by step. Arthur's Magazine was started in 1844. First for Women was started in 1989. 1844 (Arthur's Magazine) < 1989 (First for Women), so Arthur's Magazine was started first.

Answer: Arthur's Magazine

|||

Question: {{question}}

Thought: Let's think step by step.