

# LLM - projekt

Mikołaj Garbowski

[notes.mgarbowski.pl](http://notes.mgarbowski.pl) - RAG

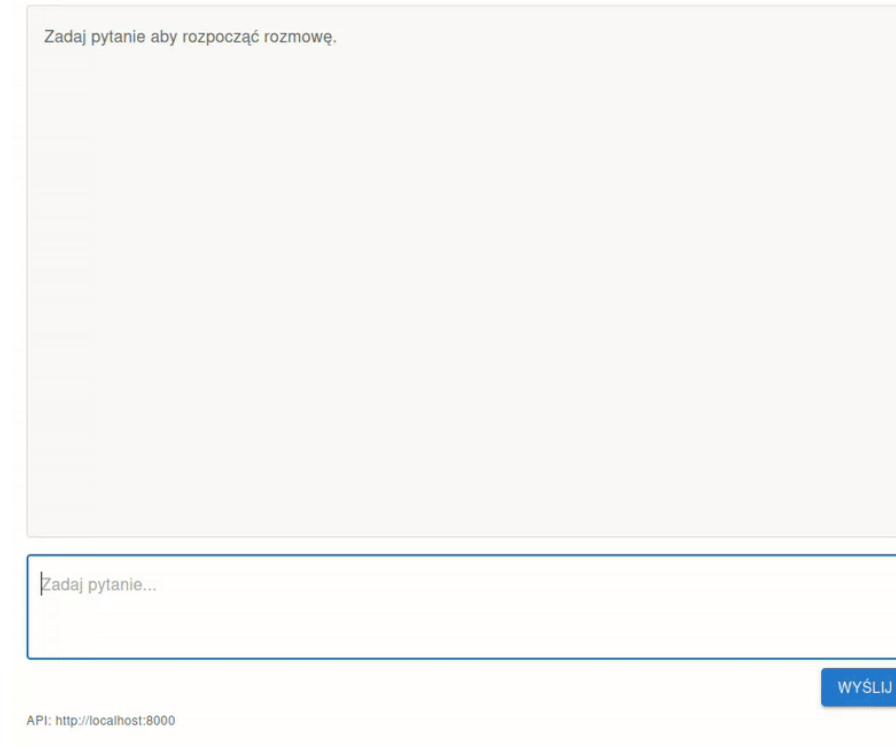


Figure 1: Interfejs użytkownika

## Opis

Celem projektu jest implementacja systemu Retrieval-Augmented Generation (RAG) wykorzystującego duży model językowy (LLM) do generowania odpowiedzi na pytania użytkownika na podstawie bazy dokumentów oraz ewaluacja skuteczności opracowanego rozwiązania na samodzielnie przygotowanym zbiorze testowym.

Za zbiór dokumentów posługują moje notatki z wykładów na Politechnice Warszawskiej w formacie markdown, sporządzone w toku studiów inżynierskich.

## Koncepcja rozwiązania

- Dokumenty zostaną podzielone na fragmenty (chunks)
  - długość tak dobrana, żeby zmieścić się w oknie kontekstowym modeli
- Fragmenty z zanurzeniami i metadanymi będą zapisane w bazie danych
- Aplikacja backend
  - FastAPI + SentenceTransformers + transformers
  - wyszukiwanie (retrieval)
    - \* leksykalne: wyszukiwanie pełnotekstowe w PostgreSQL
    - \* semantyczne: bi-enkoder do wyznaczania wektorów zanurzeń
  - reranking
    - \* wyniki wyszukiwania z obu wariantów są łączone
    - \* cross-enkoder ocenia istotność każdego z fragmentów względem pytania
    - \* wynikiem jest lista fragmentów posortowana wg. istotności
  - generacja odpowiedzi
    - \* na wejście LLM trafia pytanie wraz z kontekstem złożonym z najbardziej istotnych fragmentów
- Aplikacja frontend
  - prosta aplikacja w React z okienkiem czatu w stylu popularnych serwisów
- Baza danych
  - PostgreSQL
  - rozszerzenie pgvector do przechowywania i indeksowania wektorów zanurzeń
  - wbudowana funkcjonalność wyszukiwania pełnotekstowego
  - rozszerzony o konfigurację dla języka polskiego (zobacz sekcję Źródła)

## Modele i algorytmy

Projekt ma charakter edukacyjny i będzie uruchamiany własnej maszynie wyposażonej w GPU NVIDIA GTX 1060 6GB. Wybierając modele, kieruję się tymi ograniczeniami sprzętowymi.

- Generator - `speakleash/Bielik-1.5B-v3.0-Instruct`
- Retriever
  - PostgreSQL full-text search (wariant leksykalny)
  - bi-enkoder `sdadas/mmlw-retrieval-roberta-large`
- Reranker
  - cross-enkoder `sdadas/polish-reranker-roberta-v3`

## Zbiór testowy

Na potrzeby ewaluacji opracowałem zbiór testowy składający się z 38 pytań wraz z listą istotnych fragmentów dokumentów oraz wzorcowymi odpowiedziami.

Plik `test.jsonl` w formacie JSONL (każda linijka to osobny JSON), pojedynczy wiersz jest w formacie:

```
{  
    "query": "Co to jest partycjonowanie zakresowe i kiedy warto je stosować?",  
    "relevantDocs": [  
        "25a8cb17-d17f-4522-8243-3da0f8f6491f",  
        "decb07c0-fb6e-4fb5-80ac-520eda2521e2"  
    ],  
    "referenceAnswers": [  
        "Partycjonowanie zakresowe dzieli dane na partycje według zakresów wartości klucza..."  
    ]  
}
```

Został przygotowany z użyciem LLM GPT-5 mini i poddany ręcznej weryfikacji i korekcie.

## Ewaluacja

Oddziennie przeprowadziłem ewaluację komponentu retrieval oraz jakości generowanych odpowiedzi (pełnego potoku RAG).

### Retrieval

Metryki są wyliczane na podstawie `relevantDocs` w zbiorze testowym i wyników zwróconych przez komponent retrieval. Porównywane są trzy warianty:

- leksykalny (PostgreSQL full-text search)
- semantyczny (bi-enkoder)
- leksykalne + semantyczny + reranker

Porównywane są miary jakości Recall@k (niezależne od kolejności dokumentów w wyniku) oraz MRR@k (uwzględniające kolejność, im wyżej w rankingu jest pierwszy istotny dokument, tym lepiej).

	Fulltext Retriever (‘simple’)	Fulltext Retriever (‘polish’)	Semantic Retriever	Reranking Retriever
Recall@1	0.19	0.13	0.33	0.50
Recall@3	0.30	0.21	0.59	0.67
Recall@5	0.33	0.32	0.66	0.71
Recall@10	0.35	0.39	0.71	0.73
MRR@1	0.29	0.24	0.55	0.79
MRR@3	0.35	0.28	0.69	0.86
MRR@5	0.36	0.32	0.70	0.86
MRR@10	0.37	0.33	0.70	0.86

Zaskakujące jest, że wariant wyszukiwania leksykalnego z użyciem konfiguracji `polish` (wsparcie dla stemmingu, usuwania *stop words* itp.) wypada gorzej niż generyczny wariant `simple`. Cały zbiór dokumentów jest w języku polskim, więc spodziewałem się znacznie lepszych wyników.

## Generacja odpowiedzi

Metryki są wyliczane na podstawie `referenceAnswers` w zbiorze testowym i odpowiedzi wygenerowanych przez pełny potok RAG (z użyciem wariantu wyszukiwania leksykalnego + semantycznego + reranking).

	Prompt 1	Prompt 2
BERT Score - Precision	0.65	0.65
BERT Score - Recall	0.73	0.73
BERT Score - F1	0.68	0.69
ROUGE-L - Precision	0.13	0.12
ROUGE-L - Recall	0.36	0.34
ROUGE-L - F1	0.18	0.17
BLEU Score	3.74	1.71

Prompt 1:

*Jesteś pomocnym asystentem, odpowiadaj krótko po polsku na podstawie kontekstu. Cytuj źródła w nawiasach kwadratowych [...].*

Prompt 2 (z laboratorium):

*Odpowiedz na pytanie użytkownika wykorzystując tylko informacje znajdujące się w dokumentach, a nie wcześniejszą wiedzę. Udziel wysokiej jakości, poprawnej gramatycznie odpowiedzi w języku polskim. Odpowiedź powinna zawierać cytowania do dokumentów, z których pochodzą informacje. Zacytuj dokument za pomocą symbolu [nr\_dokumentu] powołując się na fragment np. [0] dla fragmentu z dokumentu 0. Jeżeli w dokumentach nie ma informacji potrzebnych do odpowiedzi na pytanie, zamiast odpowiedzi zwróć tekst: Nie udało mi się odnaleźć odpowiedzi na pytanie.*

## Instrukcja uruchomienia

Wymagana jest instalacja:

- Docker
- Docker Compose
- uv
- just
- Node.js i npm

Wszystkie polecenia są uruchamiane za pomocą just. Lista dostępnych poleceń można wyświetlić poleceniem just -l.

### Instalacja zależności

```
just install
```

### Tekstowy interfejs użytkownika

```
just answer "Twoje pytanie"
```

### Przygotowanie bazy danych

Surowy zbiór danych (notatki w formacie markdown) znajduje się w repozytorium (data/raw).

```
just chunk      # Podziel dokumentów na fragmenty
just index      # Wyznaczenie zanurzeń dla wszystkich fragmentów
just create_db  # Uruchomienie bazy danych w kontenerze Docker
just seed        # Załadowanie fragmentów z wyliczonymi zanurzeniami do bazy danych
```

### Uruchomienie aplikacji

Wymaga uruchomionej bazy danych

```
just devb       # backend
just devf       # frontend
```

### Ewaluacja

```
just eval-retriever  # Ewaluacja komponentu wyszukiwania
just eval-pipeline   # Ewaluacja jakości generowanych odpowiedzi
```

## Źródła

- Wykłady z przedmiotu Podstawy Wielkich Modeli Językowych na wydziale EiT PW
- <https://medium.com/@jesvinkjustin/from-zero-to-rag-the-art-of-document-chunking-and-embedding-for-rag-d9764695cc46>
- <https://medium.com/@nitinprodduturi/using-postgresql-as-a-vector-database-for-rag-retrieval-augmented-generation-c62cfbd9560>
- <https://www.evidentlyai.com/ranking-metrics/precision-recall-at-k>
- <https://www.evidentlyai.com/ranking-metrics/mean-reciprocal-rank-mrr>
- Konfiguracja PostgreSQL dla języka polskiego