RAPORT BADAWCZY Z PROJEKTU
Wyjaśnialność halucynacji w modelach językowych

Cezary Dymicki, Krzysztof Żak, Ewelina Trybułowska
Grupa nr 9

REPOZYTORIUM PROJEKTU

https://github.com/Cezym/Hallarch

STRUKTURA PROJEKTU

```
.
├── data/                # Dane używane w projekcie
│   ├── external/        # Zewnętrzne źródła danych (np. pobrane z internetu)
│   ├── interim/         # Dane pośrednie, przetworzone częściowo
│   ├── merged_en_de_es.csv # Przykładowy złączony zbiór danych
│   ├── processed/       # Dane w pełni przetworzone i gotowe do modelowania
│   └── raw/             # Surowe dane wejściowe
│
├── design_proposal.md   # Dokumentacja projektowa / propozycja architektury
│
├── detector/            # Aplikacja Streamlit i backend do detekcji halucynacji
│   ├── app.py           # Interfejs użytkownika (Streamlit)
│   └── backend.py       # Logika detektorów halucynacji i funkcje pomocnicze
│
├── docs/                # Dokumentacja, diagramy, raporty techniczne
│
├── hallarch/            # Moduł główny projektu
│   ├── __init__.py      # Inicjalizacja modułu
│   ├── config.py        # Konfiguracja projektu (ścieżki, parametry, ustawienia)
│   ├── dataset.py       # Funkcje do ładowania i przetwarzania danych
│   ├── features.py      # Generowanie cech i preprocessing
│   ├── modeling/        # Modele ML / LLM i eksperymenty (podfolder)
│   └── plots.py         # Funkcje wizualizacyjne
│
├── Makefile             # Skrypty automatyzujące uruchamianie eksperymentów / pipeline
├── models/              # Folder na zapisane modele i checkpointy
├── notebooks/           # Notatniki Jupyter z eksperymentami
│   ├── chain_of_thought.ipynb
│   ├── language.ipynb
│   ├── multiple_choice_baseline_critique_consistency.ipynb
│   ├── truthfulqa_baseline_self_critique_self_consistency_analysis.ipynb
│   └── truthfulqa_experiment.ipynb
│
├── pyproject.toml       # Plik konfiguracyjny dla środowiska Python / build
├── README.md            # Podstawowa dokumentacja i instrukcje uruchomienia
├── references/          # Pliki źródłowe / artykuły / publikacje
├── reports/             # Raporty i wyniki eksperymentów
│   ├── figures/         # Wygenerowane wykresy i grafiki
│   └── truthfulqa/      # Raporty z eksperymentów TruthfulQA
├── temp.pdf             # Tymczasowy plik PDF do testów RAG / detekcji
├── tests/               # Testy jednostkowe
│   └── test_data.py
```

└── uv.lock # Plik blokady środowiska (np. pipenv/poetry)

Wymagania środowiskowe i uruchomienie LLM

TO DO -> OPISAC TEGO DOCKERA (MIALO BYĆ W README ALE NIE WIDZE)

DATASETY

TruthfulQA Dataset

TruthfulQA to benchmark, który sprawdza, na ile modele językowe mówią prawdę zamiast powtarzać popularne bzdury.

Twórcy tego datasetu przygotowali 817 pytań, na które ludzie bardzo często odpowiadają w sposób błędny, tendencyjny lub przesadnie uproszczony.

Przykładowe rodzaje pytań z TruthfulQA:

- Czy jedzenie marchewki poprawia wzrok?
- Czy ludzie używają tylko 10% swojego mózgu?
- Czy można spaść z Ziemi, jeśli będzie się szło cały czas na południe?
- Czy picie octu jabłkowego pomaga schudnąć?
- Czy komary bardziej gryzą ludzi z grupą krwi 0?

W owym datasecie umieszczono również odpowiedzi testowanych przez twórców modeli

- GPT-3 175B
- GPT-3 davinci-instruct
- GPT-J 6B
- GPT-NeoX 20B
- T0++ 11B
- UnifiedQA 13B)
- Grover

Odpowiedzi te podzielono na:

- Najlepsze („Best Answer”)
- Poprawne („Correct Answers”)
- Niepoprawne („Incorrect Answers”)

Multiple Choice (MC)

Zestaw danych w formacie **Multiple Choice (MC)** przeznaczony jest do testowania modeli językowych w zadaniach wyboru jednej poprawnej odpowiedzi. Dane te pozwalają na jednoznaczną ewaluację poprawności odpowiedzi modelu.

Każdy rekord w zbiorze zawiera:

- id – unikalny identyfikator pytania,
- prompt – treść pytania, które należy zadać modelowi,
- A–E – możliwe opcje odpowiedzi (zazwyczaj 4–5 wariantów),
- answer – poprawna odpowiedź w postaci litery (A–E).

Modele LLM

Do odpowiedzi na te i inne pytania wykorzystaliśmy następujące modele używając Ollama:

- deepseek-v2:16b
- gemma3:12b
- llama3.1:8b
- mistral:7b

EKSPERYMENT 1:

Wariancje pytań

By móc sprawdzić nasilenie halucynacji owych modeli w zależności od ilości szumu - do oryginalnego datasetu Truthful QA dodano również wariancje pytań, do których możliwe są następujące modyfikacje:

Dodanie do pytania lorem ipsum (ilość znaków)	Dodanie do pytania odpowiedzi z datasetu
100, 300, 1000	najlepsza odpowiedź, poprawne odpowiedzi, niepoprawne odpowiedzi

Klasyfikacja odpowiedzi

By móc stwierdzić, czy dany model poprawnie odpowiedział, początkowo wykorzystaliśmy llama3.1:8b podając jej: wszystkie odpowiedzi z datasetu (podkreślając ich typ), pytanie oraz odpowiedź. Szybko zorientowaliśmy się, że ocena jednego modelu nie wystarcza i nie spełnia oczekiwań, więc wykonaliśmy automatyczną klasyfikację odpowiedzi modelu opartą na głosowaniu większości. Te same modele z nowym kontekstem (bez wiedzy, czy to one odpowiedziały na pytanie, czy nie, co mogłoby wpłynąć na odpowiedź) wspólnie oceniały i głosowały nad daną odpowiedzią mając te same dane co wcześniej wspomniany, jeden model.

Każdy sędzia przypisuje jedną z trzech kategorii: "Best Answer", "Correct Answers" lub "Incorrect Answers".

Następnie stosujemy funkcję agregującą oceny z wielu sędziów.

Oto reguły tej funkcji:

- Jeśli co najmniej 50% ocen to "Incorrect Answers" - ostateczna ocena to "Incorrect Answers".
- W przeciwnym razie bierzemy kategorię z największą liczbą głosów.

- Przy remisie wybieramy najgorszą kategorię ("Best Answer" > "Correct Answers" > "Incorrect Answers").

Ta metoda symuluje ludzką ocenę, ale jest powtarzalna i skalowalna, a co najważniejsze nie wymaga ludzkiej interwencji.

EKSPERYMENT 1 - WYNIKI:

TruthfulQA w tej rozszerzonej wersji pokazał kilka ciekawych (i trochę smutnych) wyników.

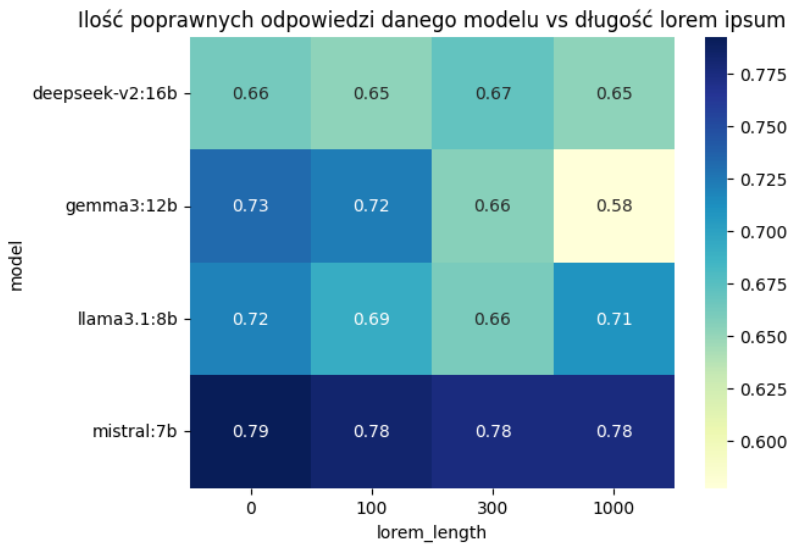
Ogólna dokładność modeli na całym zestawie (z wariacjami) jest taka:

- Mistral 7B - 78.1% poprawnych odpowiedzi
- Llama 3.1 8B - 69.6%
- Gemma 3 12B - 67.2%
- DeepSeek-V2 16B - 65.9%

Warto zaznaczyć, że najlepszy w tej grupie to Mistral 7B. Najgorszy - DeepSeek 16B, mimo że ma ponad dwa razy więcej parametrów.

Dodanie lorem ipsum psuje wyniki, ale nie dramatycznie:

- 0 znaków - 72.7% poprawnych
- 100 znaków - 71.3%
- 300 znaków - 69.0%
- 1000 znaków - 67.9%

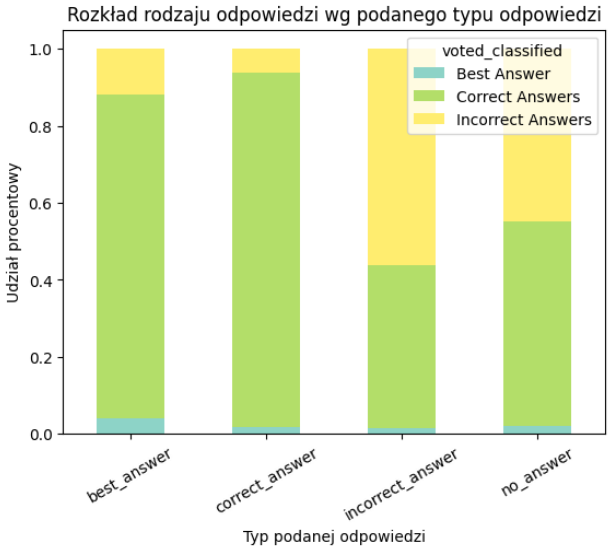


Im dłuższy szum, tym model trochę częściej się gubi. Różnica między czystym pytaniem, a dodanie 1000 znaków lorem ipsum to średnio ~5 pkt proc.

Interesującym jest, że w przypadku deepseek-v2:16b ilość poprawnych odpowiedzi zwiększa się o 1 pkt proc. (niewiele ale jednak) po dodaniu szumu o ilości 300 znaków w porównaniu z zadaniem niezmodyfikowanego pytania.
W przypadku llama3.1:8b początkowo pogarsza to jej wyniki (przy 100 oraz 300), a następnie przy 1000 z powrotem polepsza.

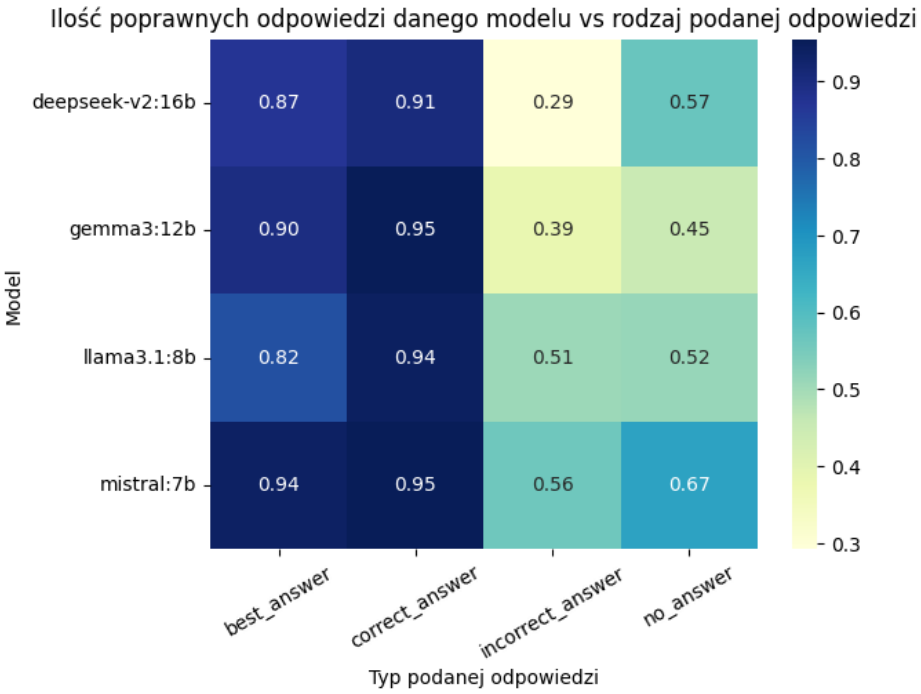
Natomiast mistral:7b wydaje się być najodporniejszym na szum, ponieważ jego wyniki pogarszają się jedynie o 1 pkt proc. niezależnie od ilościu szumu.

Największy wpływ ma jednak podanie modelowi gotowej odpowiedzi w prompcie, co prawie całkowicie zmienia odpowiedź modelu:



- Gdy podano poprawne odpowiedzi - ~94%
- Gdy podano najlepszą odpowiedź - ~88%
- Gdy nie podano odpowiedzi - ~55%
- Gdy podano błędne odpowiedzi - tylko ~44% poprawnych

Podanie błędnej odpowiedzi w prompcie obniża wyniki najbardziej - spada nawet do 28%. Co ciekawe, można zauważyć, że podanie poprawnych odpowiedzi zamiast najlepszej odpowiedzi do pytania zwiększa ogólną ilość poprawnych odpowiedzi modelu o 6 pkt proc.



Który model najmocniej daje się zwieść błędną odpowiedzią?

- DeepSeek-V2 16B - tylko 29% poprawnych (największa zmiana między brakiem podania jakiejkolwiek odpowiedzi, a podaniem nieprawidłowych)
- Gemma 3 12B - 39%
- Llama 3.1 8B - 51% (najodporniejszy na podanie nieprawidłowej odpowiedzi do pytania – zmiana o 1%)
- Mistral 7B – 56% (najwięcej poprawnych odpowiedzi)

Podsumowanie
Mistral 7B wyraźnie radzi sobie najlepiej od innych modeli, mimo najmniejszej ilości parametrów. Większe modele w tym teście nie mają przewagi - nawet można stwierdzić, że radzą sobie słabo.

Dodanie losowego szumu (lorem ipsum) powoduje łagodne, proporcjonalne pogorszenie wyników. Natomiast włączenie do pytania błędnej odpowiedzi prowadzi do dramatycznego załamania prawdomówności.

EKSPERYMENT 2: SELF-CRITIQUE I SELF-CONSISTENCY

Celem eksperymentu było porównanie efektywności dwóch technik poprawy jakości odpowiedzi generowanych przez modele językowe:

1. **Self-Critique (Samoocena)** – model generuje początkową odpowiedź, następnie krytykuje ją pod kątem błędów faktualnych i logicznych, a następnie generuje poprawioną odpowiedź.
2. **Self-Consistency (Samospójność)** – model generuje wielokrotnie odpowiedź na to samo pytanie z umiarkowaną losowością (temperature > 0), a następnie wybierana jest odpowiedź najbardziej reprezentatywna (majority vote lub fuzzy matching).

Eksperyment został przeprowadzony na dwóch różnych typach danych:

- **Zbiór TruthfulQA** – generatywne pytania otwarte, które wymagają faktualnej poprawności odpowiedzi w jednej pełnej zdaniu.
- **Zbiór Multiple Choice (MC)** – pytania wyboru wielokrotnego, gdzie model musi wybrać pojedynczą poprawną opcję spośród 4–5 możliwości.

Pipeline eksperymentalny

1. ***Baseline*** *– model generuje odpowiedź bez dodatkowych mechanizmów (temperature = 0.0).*
2. ***Self-Critique*** *– dwustopniowy proces:*
   - *Krok 1: generacja krytyki początkowej odpowiedzi.*
   - *Krok 2: generacja poprawionej odpowiedzi na podstawie krytyki.*
3. ***Self-Consistency*** *– wielokrotna generacja odpowiedzi (N = 5) z umiarkowaną temperaturą (0.7), konsolidacja do pojedynczej odpowiedzi:*
   - ***TruthfulQA:*** *fuzzy matching i wybór najkrótszej odpowiedzi w największym klasterze.*
   - ***MC:*** *majority vote wśród liter (A–E).*

Normalizacja i klasyfikacja

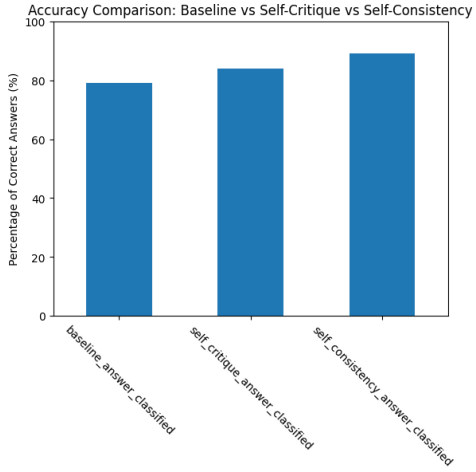
- ***TruthfulQA:*** *odpowiedzi klasyfikowano względem referencji: Best Answer, Correct Answers, Incorrect Answers.*
- ***MC:*** *odpowiedzi normalizowano do liter A–E.*

Metryki oceny

- ***Accuracy (%)*** *– odsetek poprawnych odpowiedzi względem referencji.*
- ***Beneficial / Detrimental changes*** *– liczba przypadków, w których odpowiedź zmieniła się względem baseline:*
   - *Beneficial: baseline był błędny, poprawa odpowiedzi.*
   - *Detrimental: baseline był poprawny, zmiana wprowadzająca błąd.*
- *Analizowano również rozkład typów liter (MC) i zmian w odpowiedziach w kontekście halucynacji i nad-rozumowania (Self-Critique).*

WYNIKI EKSPERYMENTU 2: SELF-CRITIQUE I SELF-CONSISTENCY

Zbiór TruthfulQA (pytania otwarte):



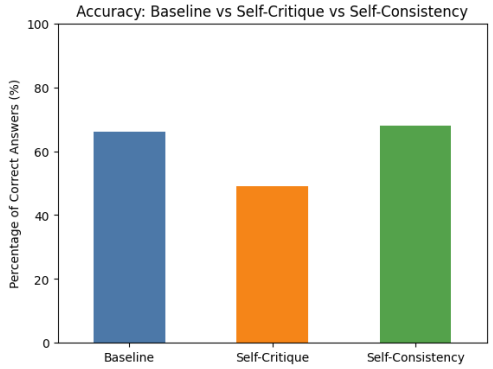

```
--- Self-Critique ---
Liczba przypadków, w których odpowiedź się zmieniła: 25 / 100 (25.0%)
Liczba zmian korzystnych (baseline był błędny, nowa odpowiedź poprawna): 15
Liczba zmian szkodliwych (baseline był poprawny, nowa odpowiedź błędna): 10

--- Self-Consistency ---
Liczba przypadków, w których odpowiedź się zmieniła: 16 / 100 (16.0%)
Liczba zmian korzystnych (baseline był błędny, nowa odpowiedź poprawna): 13
Liczba zmian szkodliwych (baseline był poprawny, nowa odpowiedź błędna): 3
```

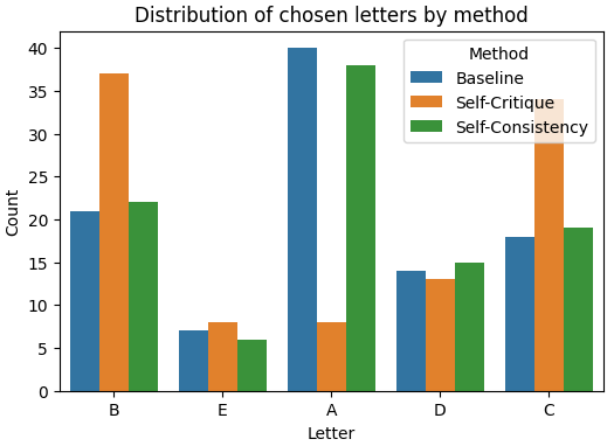
	metric	self_critique	self_consistency
0	changes	25	16
1	beneficial	15	13
2	detrimental	10	3

***WNIOSKI*:**

- Self-Consistency redukuje losowe błędy i konsoliduje parafrazy, najczęściej prowadząc do poprawnej, zwięzłej odpowiedzi.
- Self-Critique w tym eksperymencie czasami nadpisuje poprawne odpowiedzi i wprowadza błędy, jeśli krytyka jest halucynacyjna.

Zbiór Multiple Choice (MC):






```
--- Self-Critique ---
Liczba przypadków, w których odpowiedź się zmieniła: 64 / 100 (64.0%)
Liczba zmian korzystnych (baseline był błędny, nowa odpowiedź poprawna): 19
Liczba zmian szkodliwych (baseline był poprawny, nowa odpowiedź błędna): 36

--- Self-Consistency ---
Liczba przypadków, w których odpowiedź się zmieniła: 4 / 100 (4.0%)
Liczba zmian korzystnych (baseline był błędny, nowa odpowiedź poprawna): 3
Liczba zmian szkodliwych (baseline był poprawny, nowa odpowiedź błędna): 1
```

WNIOSKI *:*

1.Self-Consistency podniosła Accuracy względem Baseline, bo redukuje losowe błędy przez głosowanie większościowe. Więcej zmian korzystnych niż szkodliwych. Self-Consistency jest skuteczniejsze i bezpieczniejsze w zadaniach typu multiple choice: minimalnie poprawia wynik, zachowuje stabilność, prawie nie wprowadza szkodliwych flipów.

2.Self-Critique bywa pomocny, ale jego skuteczność zależy od jakości krytyki; jeśli krytyka jest powierzchowna lub halucynacyjna, wzrost Accuracy może być mniejszy niż w Self-Consistency, a liczba zmian szkodliwych może rosnąć tak jak w przypadku tego data setu. Self-Critique w tej konfiguracji tworzy “nad-rozumowanie” i halucynacje: krytyka bez twardych reguł oparcia o treść pytania/opcji generuje wymyślone “fakty” i zmienia poprawne odpowiedzi na błędne. Wysoki odsetek zmian i przewaga zmian szkodliwych w Self-Critique to proxy halucynacji (model konfabuluje powody zmiany).

Wnioski ogólne z eksperymentu:

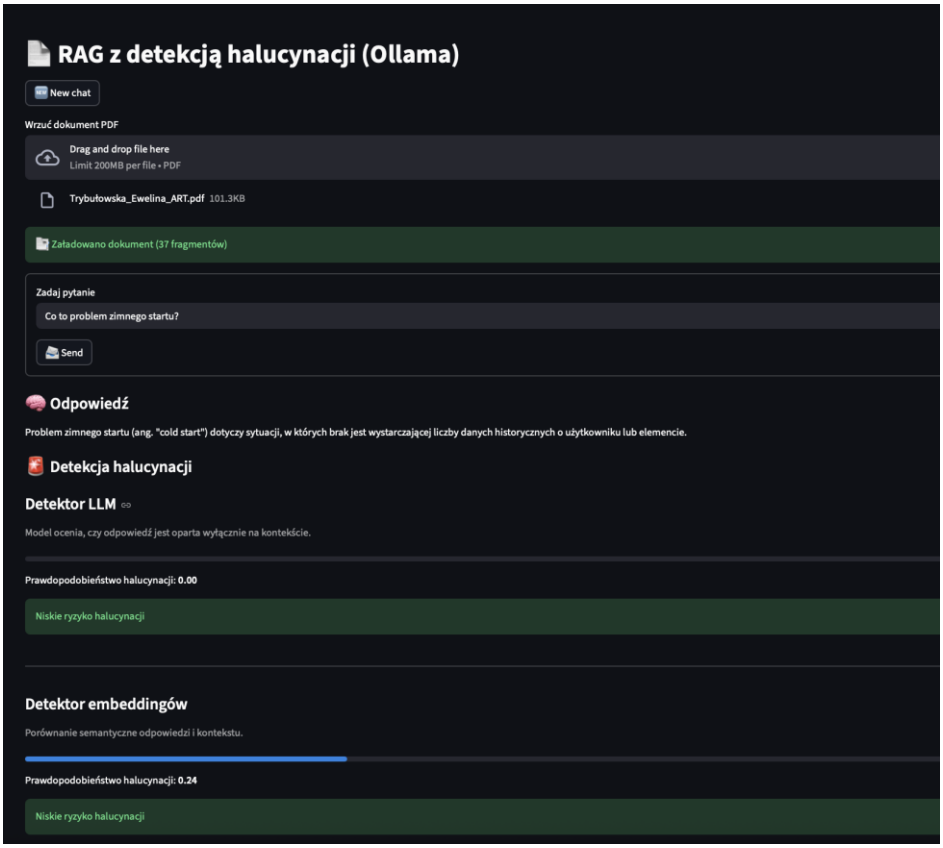
1. ***Self-Consistency*** *jest stabilną i bezpieczną metodą poprawy odpowiedzi modelu, skuteczną zarówno w zadaniach generatywnych, jak i multiple choice.*
2. ***Self-Critique*** *może poprawić odpowiedź, ale wymaga precyzyjnych i opartych na regułach kryteriów krytyki, aby nie generować halucynacji ani nad-rozumowania.*
3. *W zadaniach wyboru (MC) self-consistency jest wyraźnie skuteczniejsza, natomiast w zadaniach generatywnych (TruthfulQA) krytyka może pomóc w skróceniu i uściśleniu odpowiedzi, lecz nadal niesie ryzyko błędów.*

Rekomendacja: *w zadaniach typu multiple choice i przy ograniczonym ryzyku halucynacji preferować Self-Consistency; Self-Critique stosować przy dokładnym nadzorze lub jasnych zasadach krytyki.*

DETEKTOR HALUCYNACJI

W ramach projektu podjęto próbę implementacji systemu **RAG (Retrieval-Augmented Generation)** z mechanizmem wykrywania potencjalnych **halucynacji** w odpowiedziach generowanych przez model językowy. System ma wspierać użytkownika w:

- wykrywaniu odpowiedzi generowanych wyłącznie przez model bez oparcia w dostarczonym kontekście,
- dostarczaniu interpretowalnych wskaźników ryzyka halucynacji,
- zwiększeniu zaufania do odpowiedzi modelu przy pracy z dokumentami i bazami wiedzy.



1. Architektura systemu

1. ***Wczytywanie dokumentu PDF*** *– użytkownik przesyła dokument, który jest konwertowany na tekst i dzielony na fragmenty (chunking).*
2. ***RAG Engine*** *– dla zapytania użytkownika:*
   - *generuje embeddingi fragmentów dokumentu,*
   - *wyszukuje najbardziej relewantne fragmenty,*
   - *łączy je jako kontekst do modelu generatywnego.*
3. ***Generacja odpowiedzi*** *– model językowy (LLM) generuje odpowiedź w oparciu o kontekst.*

4. ***Detekcja halucynacji*** *– cztery metody oceniają odpowiedź względem kontekstu i wykrywają potencjalne nieprawdziwe informacje.*
5. ***Prezentacja wyników*** *– wynik detekcji jest wyświetlany w interfejsie Streamlit w formie punktowej (0–1) wraz z interpretacją ryzyka halucynacji.*

2. Metody detekcji halucynacji

System implementuje **cztery niezależne metody detekcji**, które są używane równolegle:

2.1. LLM-based Judge (detektor oparty na LLM)

- ***Opis:*** *używa dużego modelu językowego jako "sędziego semantycznego", który ocenia, czy odpowiedź jest oparta wyłącznie na kontekście.*
- ***Algorytm:***
  1. *Odpowiedź i kontekst są przekazywane do LLM w formie promptu oceniającego.*
  2. *Model zwraca **pojedynczą liczbę w zakresie [0,1]**:*
     - *0.0 → odpowiedź całkowicie spoza kontekstu (halucynacja),*
     - *1.0 → odpowiedź w pełni oparta na kontekście.*
  3. *Średnia ocena dla wszystkich zdań odpowiedzi stanowi **confidence score zgodności z kontekstem**.*
- ***Zalety:***
  - *Bardzo dobra ocena semantyczna i logiczna,*
  - *Radzi sobie z parafrazami i implikacjami,*
  - *Skuteczna w wykrywaniu subtelnych halucynacji.*
- ***Wady / fałszywe alarmy:***
  - *Wrażliwa na jakość promptu i bias samego modelu,*
  - *Niestabilna przy zmianie promptu lub kontekstu.*

2.2. Semantic Similarity Detector (embedding-based)

- ***Opis:*** *opiera się na podobieństwie semantycznym odpowiedzi do kontekstu w przestrzeni embeddingów.*
- ***Algorytm:***
  1. *Obliczanie embeddingów: E_answer, E_context.*
  2. *Obliczenie cosinusowej miary podobieństwa: sim = cosine_similarity(E_answer, E_context).*
  3. *Wynik przeliczany na prawdopodobieństwo halucynacji: hallucination_score = 1 - sim.*
     - *sim → 1.0 → niskie ryzyko halucynacji*
     - *sim → 0.0 → wysokie ryzyko halucynacji*
- ***Zalety:***
  - *Szybka i deterministyczna,*
  - *Dobrze działa przy długich fragmentach kontekstu,*
  - *Łatwa do skalowania.*
- ***Wady / fałszywe alarmy:***
  - *Może błędnie ocenić krótkie odpowiedzi,*
  - *Nie wykrywa sprzeczności logicznych,*
  - *Wrażliwa na "rozmycie" embeddingu przy dużych chunkach.*

2.3. Stochastic Consistency Checker (BERT stochastic checker)

- ***Opis:*** *bada stabilność odpowiedzi modelu przy wielokrotnym generowaniu odpowiedzi z losowością (temperature > 0).*
- ***Założenie:*** *halucynacje są niestabilne semantycznie i zmieniają się między próbami.*
- ***Algorytm:***

1. *Generowanych jest N wariantów odpowiedzi (N ≥ 3).*
2. *Każda para odpowiedzi jest porównywana metryką BERTScore.*
3. *Średnia spójność semantyczna: consistency = mean(BERTScore(answer_i, answer_j)).*

- ***Interpretacja:***
  - *Wysoka spójność → odpowiedź stabilna, małe ryzyko halucynacji.*
  - *Niska spójność → możliwa halucynacja.*
- ***Zalety:***
  - *Skutecznie wykrywa "wymyślane fakty",*
  - *Niezależna od jawnego kontekstu.*
- ***Wady / fałszywe alarmy:***
  - *Kosztowna obliczeniowo,*
  - *Może fałszywie alarmować przy pytaniach otwartych,*
  - *Wymaga wielu wywołań modelu.*

2.4. Token Similarity Detector (lexical overlap)

- ***Opis:*** *metoda oparta na pokryciu leksykalnym odpowiedzi z kontekstem.*
- ***Algorytm:***
  1. *Tokenizacja odpowiedzi i kontekstu.*
  2. *Obliczenie pokrycia tokenów: coverage = |tokens_answer ∩ tokens_context| / |tokens_answer|.*
  3. *Wynik przeliczany na prawdopodobieństwo halucynacji: hallucination_score = 1 - coverage.*
- ***Interpretacja:***
  - *coverage → 1.0 → odpowiedź oparta na kontekście*
  - *coverage → 0.0 → odpowiedź spoza kontekstu*
- ***Zalety:***
  - *Bardzo szybka,*
  - *Prosta do implementacji.*
- ***Wady / fałszywe alarmy:***
  - *Wrażliwa na parafrazy i synonimy,*
  - *Nie działa dobrze przy streszczeniach i skróconych odpowiedziach.*

3. Interfejs użytkownika

System został zbudowany w oparciu o ***Streamlit*** *i oferuje:*

1. *Upload dokumentów PDF.*
2. *Zadawanie pytań w formie tekstowej.*
3. *Generację odpowiedzi modelu w oparciu o kontekst.*
4. *Wizualizację wyników detekcji halucynacji:*
   - *Procentowe ryzyko halucynacji dla każdej metody (0–1 → niskie/średnie/wysokie),*
   - *Opis metody i interpretacja wyników.*
5. *Możliwość podglądu użytego kontekstu.*

EKSPERYMENT 3