Laboratorium 5 Analiza zjawiska halucynacji w modelach LLM

1. Wprowadzenie

Halucynacje (ang. hallucinations) to zjawisko, w którym model językowy generuje tekst, który jest płynny gramatycznie i semantycznie, ale faktograficznie nieprawdziwy lub niemający oparcia w danych wejściowych. Jest to jeden z kluczowych problemów bezpieczeństwa i niezawodności systemów opartych na LLM.

Rodzaje halucynacji:

- Halucynacje faktograficzne — model podaje nieprawdziwe fakty (np. błędne daty, zmyślone bibliografie, nieprawdziwe cytaty)
- Halucynacje kontekstowe — model odpowiada sprzecznie z podanym kontekstem (Faithfulness errors)
- Halucynacje konfabulacyjne — model uzupełnia luki w wiedzy zmyślonymi, ale spójnymi informacjami
- Halucynacje tożsamości — model myli osoby, przypisuje im błędne osiągnięcia

Przyczyny halucynacji:

- Trening na danych zawierających błędy lub sprzeczności
- Mechanizm próbkowania (sampling) wprowadzający losowość
- Dążenie modelu do spójności językowej kosztem dokładności faktycznej
- Brak wbudowanego mechanizmu weryfikacji faktów
- Efekt sykofancja (sycophancy) — model potwierdza błędne założenia użytkownika

> Ważne: Badanie halucynacji jest aktywną dziedziną badań NLP. Biblioteki takie jak RAGAS, TruLens i DeepEval pozwalają na automatyczną detekcję halucynacji kontekstowych.

2. Przykłady implementacji

Przykład 1: Prowokowanie i dokumentowanie halucynacji

```
from transformers import pipeline

gen = pipeline('text-generation', model='gpt2-medium')

# Prompty podatne na halucynacje
hallucination_prompts = [
    'The Nobel Prize in Physics 2024 was awarded to',
    'Albert Einstein died in the year',
    'The population of Warsaw according to 2023 census is',
    'The main author of the Python programming language was',
    'According to recent studies, coffee consumption',
```

```
]

print('=== Analiza halucynacji ===')
for prompt in hallucination_prompts:
    result = gen(prompt, max_new_tokens=40, do_sample=True, temperature=0.7)
    full   = result[0]['generated_text']
    completion = full[len(prompt):]
    print(f'Prompt:     {prompt}')
    print(f'Completion: {completion.strip()[:80]}')
    print(f'--- Zweryfikuj faktyczną poprawność powyższego! ---')
    print()
```

Przykład 2: Detekcja halucynacji kontekstowych (NLI)

```
# Natural Language Inference do wykrywania niespójności
from transformers import pipeline

nli = pipeline('text-classification',
               model='cross-encoder/nli-deberta-v3-small')

def check_faithfulness(context, claim):
    '''Sprawdza czy claim wynika z context (entailment).'''
    result = nli(f'{context} [SEP] {claim}')[0]
    return result['label'], result['score']

context = ('Warsaw is the capital of Poland. It has a population of 1.8 million. '
           'The city was founded in the 13th century.')

claims = [
    'Warsaw is the capital of Poland.',           # prawda
    'Warsaw has 5 million inhabitants.',          # sprzeczność
    'The city was established in the 1300s.',     # parafra (prawda)
    'Warsaw is located on the Vistula River.',    # prawda, brak w kontekscie
]

for claim in claims:
    label, score = check_faithfulness(context, claim)
    print(f'{label:15s} ({score:.3f})  |  {claim}')
```

Przykład 3: Porównanie pewności modelu (confidence)

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
import numpy as np

tokenizer = AutoTokenizer.from_pretrained('gpt2')
```

```
model     = AutoModelForCausalLM.from_pretrained('gpt2')
model.eval()

def get_generation_confidence(prompt, answer):
    '''Oblicza średnie prawdopodobieństwo tokenów w odpowiedzi.'''
    full_text  = prompt + answer
    input_ids  = tokenizer.encode(full_text, return_tensors='pt')
    prompt_len = len(tokenizer.encode(prompt))

    with torch.no_grad():
        outputs = model(input_ids, labels=input_ids)
        logits  = outputs.logits

    probs     = torch.softmax(logits, dim=-1)
    token_ids = input_ids[0][prompt_len:]
    token_probs = [probs[0, prompt_len-1+i, tid].item()
                   for i, tid in enumerate(token_ids)]
    return np.mean(token_probs) if token_probs else 0.0

prompt = 'The capital of France is'
answers = [' Paris.', ' Berlin.', ' London.', ' Rome.']
for ans in answers:
    conf = get_generation_confidence(prompt, ans)
    print(f'Confidence={conf:.4f}  |  ...{ans}')
```

Przykład 4: Techniki mitygacji (łagodzenia) halucynacji

```
from transformers import pipeline

gen = pipeline('text-generation', model='gpt2')

def generate_with_consistency_check(prompt, n=5, threshold=0.6):
    '''Self-consistency: generuje N odpowiedzi i wybiera najczęstszą.'''
    responses = []
    for _ in range(n):
        r = gen(prompt, max_new_tokens=30, do_sample=True, temperature=0.7)
        responses.append(r[0]['generated_text'][len(prompt):].strip())

    # Grupowanie podobnych odpowiedzi (uproszczone: po pierwszych 5 słowach)
    groups = {}
    for r in responses:
        key = ' '.join(r.split()[:5])
        groups[key] = groups.get(key, 0) + 1

    best_key = max(groups, key=groups.get)
    confidence = groups[best_key] / n
```

```
    best_answer = [r for r in responses
                   if ' '.join(r.split()[:5]) == best_key][0]

    return best_answer, confidence, groups

prompt = 'Albert Einstein was born in the city of'
answer, conf, groups = generate_with_consistency_check(prompt, n=5)
print(f'Best answer (confidence={conf:.0%}): {answer[:60]}')
print(f'Distribution: {groups}')
```

3. Zadania do samodzielnego rozwiązania

Zadanie 5.1: Katalog halucynacji

Stwórz zestaw 15 promptów zaprojektowanych do wywoływania różnych typów halucynacji (faktograficzne, konfabulacyjne, tożsamościowe). Wygeneruj odpowiedzi GPT-2. Każdą odpowiedź zweryfikuj i sklasyfikuj: (a) prawdziwa, (b) halucynacja częściowa, (c) halucynacja pełna. Oblicz wskaźnik halucynacji. Czy pewne typy pytań są bardziej podatne na halucynacje?

Zadanie 5.2: Porównanie faithfulness modeli QA

Przygotuj 10 par (kontekst, pytanie) gdzie kontekst zawiera celowo błędne informacje (np. 'Einstein urodził się w 1900 roku'). Sprawdź, czy model QA (BERT) i generatywny (FLAN-T5) powtarzają te błędy, czy poprawiają je na podstawie własnej wiedzy. Który model jest bardziej 'wierny' kontekstowi? Który jest bardziej bezpieczny w zastosowaniach produkcyjnych?

Zadanie 5.3: Implementacja detektora halucynacji

Używając modelu NLI (przykład 2), zbuduj funkcję hallucination_score(context, generated_text), która dzieli wygenerowany tekst na zdania (nltk.sent_tokenize), sprawdza każde zdanie na entailment/contradiction względem kontekstu i zwraca: procent zdań sprzecznych, procent entailment, procent neutral. Przetestuj na 5 dokumentach z wygenerowanymi streszczeniami.

Zadanie 5.4: Strategia mitygacji halucynacji

Zaimplementuj i porównaj 3 strategie ograniczania halucynacji: (1) obniżenie temperature do 0.3, (2) dodanie 'Answer only based on the provided text:' do promptu, (3) self-consistency voting (N=5 odpowiedzi, większościowe głosowanie). Przetestuj każdą strategię na tym samym zestawie 5 pytań wymagających precyzji faktycznej. Oceń efektywność każdej metody używając NLI jako miernika faithfulness.