

***Wojskowa Akademia Techniczna
im. Jarosława Dąbrowskiego***



Wydział Cybernetyki, kierunek informatyka - inżynieria systemów

Sprawozdanie z projektu z przedmiotu:

Standarty w projektowaniu systemów dialogowych

Temat projektu:

Techniki inżynierii odpowiedzi (prompt engineering), przetwarzanie języka naturalnego poleceń dla modeli AI

Opracował: Radosław Relidzyński, Grupa: WCY23IX3S4

Spis treści

Wstęp.....	3
Wykorzystywana technologia.....	3
Analizowane techniki	3
Least-to-Most Prompting	3
Self-Asking Prompting	4
Meta-Prompting	4
Chain-Of-Thought Prompting	4
ReAct	4
Testowe problemy	4
Implementacja	5
Obsługa zapytań do API	6
Klasa bazowa dla problemów.....	7
Klasa dla problemów wzrostu	8
Klasa dla problemów czasu	10
Klasa dla problemów liter	12
Uruchamianie sprawdzania	14
Analiza wyników	14
Podsumowanie	15

Wstęp

Zakres projektu

Zadaniem projektu jest zaimplementować różnego rodzaju techniki inżynierii odpowiedzi i sprawdzić jaki mają one wpływ na jakość odpowiedzi modelu językowego.

Cel projektu

W ramach zadania celem jest sprawdzenie jaką skuteczność mają dane techniki i stworzyć przestrzeń do wywnioskowania jaką one mają skuteczność

Wykorzystywana technologia

Projekt zostanie utworzony w języku python w IDE „PyCharm”

Wykorzystana zostanie biblioteka „openai” to tworzenia połączenia z API dla Chata GPT od OpenAI.

OpenAI – amerykańskie laboratorium badawcze założone w 2015 roku, specjalizujące się w dziedzinie sztucznej inteligencji. Organizacja składa się z dwóch części: OpenAI Incorporated (non-profit) oraz OpenAI Limited Partnership (spółka-córka, działająca w celach komercyjnych). Celem badań prowadzonych przez OpenAI jest promowanie i rozwijanie sztucznej inteligencji przyjaznej dla człowieka.

[Źródło: <https://pl.wikipedia.org/wiki/OpenAI>]

ChatGPT – chatbot opracowany przez OpenAI, wykorzystujący model GPT i służący do generowania odpowiedzi na dane wprowadzane przez użytkownika. Model ten został opracowany na podstawie dużych zbiorów danych, tak aby mógł prowadzić rozmowę i angażować się w różne tematy, od ogólnych rozmów po określone obszary wiedzy.

[Źródło: <https://pl.wikipedia.org/wiki/ChatGPT>]

Testowanym modelem będzie „**gpt-3.5-turbo**”, ponieważ jest on najbardziej podstawowym modelem GPT-3.5 pod względem mocy obliczeniowej i dokładności. Sprawia to, że jest on bardziej podatny na zwrócenie błędnego wyniku, co pozwoli na lepsze sprawdzenie działania technik.

Analizowane techniki

Least-to-Most Prompting

Ta technika polega na formułowaniu zapytań od najprostszego do najbardziej skomplikowanego. Na początku zadaje się pytania ogólne, a następnie przechodzi do

bardziej szczegółowych, co pozwala na stopniowe uzyskiwanie coraz bardziej precyzyjnych informacji.

Self-Asking Prompting

Polega na tym, że model samodzielnie generuje pytania pomocnicze, które pomagają mu lepiej zrozumieć problem lub zadanie. Dzięki temu technika ta pozwala na bardziej dogłębną analizę tematu i uzyskiwanie dokładniejszych odpowiedzi.

Meta-Prompting

Technika ta wykorzystuje wiedzę o samym modelu oraz jego sposobie działania do optymalizacji zapytań. Umożliwia to tworzenie bardziej efektywnych promptów, które lepiej wykorzystują możliwości modelu, prowadząc do bardziej trafnych odpowiedzi.

Chain-Of-Thought Prompting

Polega na prowadzeniu modelu przez ciąg myślowy krok po kroku, co pomaga w rozwiązywaniu złożonych problemów. Każdy krok jest dokładnie opisywany, co umożliwia modelowi lepsze zrozumienie procesu myślowego i generowanie bardziej spójnych oraz logicznych odpowiedzi.

ReAct

Ta technika łączy działanie (ang. "Action") z refleksją (ang. "Reflection"), umożliwiając modelowi na interaktywne rozwiązywanie problemów. Model najpierw podejmuje działanie, a następnie analizuje swoje wyniki, co pozwala na poprawianie błędów i udoskonalanie odpowiedzi w czasie rzeczywistym.

Źródła informacji o technikach

- <https://www.promptingguide.ai/techniques>
- <https://cobusgreyling.medium.com/12-prompt-engineering-techniques-644481c857aa>
- <https://thestory.is/en/journal/prompt-engineering/>
- <https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-prompt-engineering/>

Testowe problemy

Poniżej przedstawione jest uruchomienie dla 10 różnych problemów, aby sprawdzić jak Chat GPT radzi sobie bez zastosowania jakiejkolwiek techniki. Błędne odpowiedzi są podkreślone na czerwono, poprawne na zielono.

Problem: Jacek jest wyższy od Tomka, ale niższy od Adama. Adam jest niższy od Bartka. Kto jest najniższy?

Response: Najniższy jest Adam.

Solution: Tomek

Problem: W układance 5x5 brakuje jednego kawałka w prawym dolnym rogu. Jakie są współrzędne brakującego kawałka?

Response: W przypadku układanki 5x5 brakujący kawałek znajduje się w prawym dolnym rogu, czyli na współrzędnych (5,5).

Solution: (5, 5)

Problem: Znajdź następną liczbę w sekwencji: 2, 6, 12, 20, 30.

Response: Następną liczbą w sekwencji jest 42.

Sposób rozwiązania:

$$2 + 4 = 6$$

$$6 + 6 = 12$$

$$12 + 8 = 20$$

$$20 + 10 = 30$$

$$30 + 12 = 42$$

Solution: 42

Problem: Jaki jest wynik $1 + 1 \times 0 + 1$?

Response: Wynik jest równy 2.

$$1 + 1 = 2$$

$$2 \times 0 = 0$$

$$0 + 1 = 1$$

Więc $1 + 1 \times 0 + 1 = 2$.

Solution: 2

Problem: Jeśli dziś jest środa, to który dzień tygodnia będzie za 100 dni?

Response: Będzie to niedziela.

Solution: Piątek

Problem: Ułóż litery „EALNG” w kolejności alfabetycznej.

Response: AELGN

Solution: A, E, G, L, N

Problem: Jacek ma 4 jabłka, a Ania 8. Ile jabłek musi dać Ania Jackowi, aby oboje mieli tyle samo?

Response: Ania musiałaby dać Jackowi 2 jabłka, aby oboje mieli tyle samo - po 6 jabłek każdy.

Solution: 2 jabłka

Problem: Jaki kolor powstanie po zmieszaniu niebieskiego i żółtego?

Response: Po zmieszaniu niebieskiego i żółtego powstanie kolor zielony.

Solution: Zielony

Problem: Mam dwie monety, które w sumie dają 30 groszy. Jedna z nich nie jest 10-groszówką. Jakie to monety?

Response: Monety to 20-groszówka i 10-groszówka.

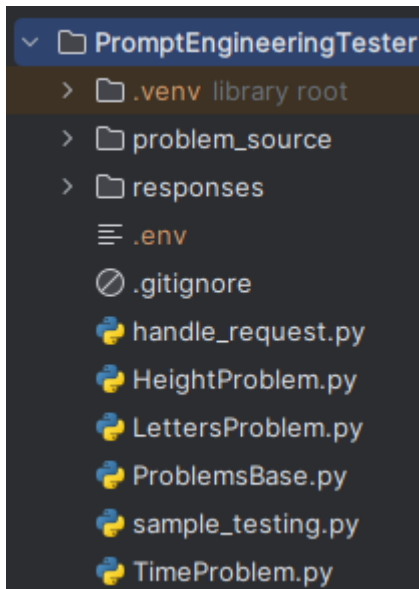
Solution: 20 groszy i 10 groszy (bo jedna z nich nie jest 10-groszówką, ale druga może być)

Jak widać na 9 zadanych problemów 3 z nich rozwiązał nieprawidłowo, posłużą one za problemy, na podstawie których przetestowane zostaną techniki pod względem skuteczności.

Implementacja

Struktura projektu

Tak wygląda struktura projektu:



Wykaz plików:

- problem_source – folder z plikami json zawierającymi problemy i ich poprawne rozwiązania (zbiór danych)
- responses – folder z folderami zawierającymi pliki tekstowe, każdy zawiera zestaw rozwiązań modelu językowego problemów danego typu dla danej techniki
- .env – plik z danymi do poprawnego połączenia się z modelem językowym
- handle_request.py – skrypt realizujący połączenie z chatem GPT
- sample_testing.py – skrypt uruchamiający różnego rodzaju problemy
- HeightProblem.py – klasa uruchamiająca problemy wzrostu dla różnych technik
- TimeProblem.py – klasa uruchamiająca problemy czasu dla różnych technik
- LettersProblem.py – klasa uruchamiająca problemy liter dla różnych technik

Poniżej prezentacja klas, a w ramach nich można zobaczyć realizację każdej techniki w zależności od problemu.

Obsługa zapytań do API

```
import os

from dotenv import load_dotenv
from openai import OpenAI

load_dotenv()

def handle_request(prompt):
    """ handle API request for AI """

    client = OpenAI(api_key=os.getenv('API_KEY'))

    chat_completion = client.chat.completions.create(
        messages=[{'role': 'user', 'content': prompt}],
        model=os.getenv('AI_MODEL'),
    )
```

```

        response = chat_completion.choices[0].message.content
        return response

if __name__ == '__main__':
    my_prompt = "What is the capital of France?"
    my_response = handle_request(my_prompt)
    print(my_response)

```

Klasa bazowa dla problemów

```

import json
import os
from abc import ABC, abstractmethod

from handle_request import handle_request

class ProblemsBase(ABC):
    """ Base class for problems to be solved by different prompt engineering
    techniques """

    def __init__(self, json_file):
        self.problems_file = os.path.join("problem_source", json_file)
        with open(self.problems_file, encoding='utf-8') as file:
            self.problems = json.load(file)

    def run_technique(self, learning_prompt):
        """ run all problems for technique """

        for problem in self.problems:
            formatted_prompt = learning_prompt % problem.get("problem")
            print(f"Problem: {problem['problem']}")
            response = handle_request(formatted_prompt)
            print(f"Response: {response}")
            print(f"Solution: {problem.get('solution')}")
            print()

    @abstractmethod
    def least_to_most_technique(self):
        """
        Least-to-most technique:
        This method involves starting with the simplest or easiest problems and
        progressively moving to more complex ones.
        It helps build confidence and understanding incrementally.
        """
        pass

    @abstractmethod
    def self_asking_technique(self):
        """
        Self-asking technique:
        This method encourages learners to ask themselves questions about the
        material they are studying.
        It helps deepen understanding and promotes active engagement with the
        content.
        """
        pass

    @abstractmethod
    def meta_technique(self):
        """
        Meta technique:
        This method involves thinking about one's own thinking processes.
        It includes strategies like planning, monitoring, and evaluating one's

```

```

understanding and learning strategies.
    """
    pass

    @abstractmethod
    def chain_of_thought_technique(self):
        """
        Chain-of-thought technique:
        This method involves linking ideas together in a logical sequence to solve
        problems or understand concepts.
        It encourages coherent and structured thinking.
        """
        pass

    @abstractmethod
    def re_act_technique(self):
        """
        ReAct technique:
        This method combines reflection and action, encouraging learners to reflect
        on their actions and adjust their strategies accordingly.
        It fosters continuous improvement and adaptive learning.
        """
        pass

```

Klasa dla problemów wzrostu

```

from ProblemsBase import ProblemsBase

class HeightProblem(ProblemsBase):
    """ HeightProblem class for height problems to be solved by different prompt
    engineering techniques """

    def __init__(self):
        super(HeightProblem, self).__init__("height_problems.json")

    def least_to_most_technique(self):
        """ Least-to-most technique for HeightProblem """

        learning_prompt = """
Jacek jest wyższy od Tomka, ale niższy od Adama. Adam jest niższy od Bartka. Kto jest
najniższy?

Rozkładamy problem na małe problemy:
Pytanie: Kto jest wyższy, Jacek czy Tomek?
Odpowiedź: Jacek
Pytanie: Kto jest wyższy, Adam czy Jacek?
Odpowiedź: Adam
Pytanie: Kto jest wyższy, Adam czy Bartek?
Odpowiedź: Bartek
Pytanie: Kto jest najniższy spośród Jacka, Tomka, Adama i Bartka?
Odpowiedź: Tomek
Poprawna odpowiedź to: Tomek

Wykorzystaj to podejście do rozwiązania tego problemu:
%s
        """

        self.run_technique(learning_prompt)

    def self_asking_technique(self):
        """ Self-asking technique for HeightProblem """

        learning_prompt = """
Jacek jest wyższy od Tomka, ale niższy od Adama. Adam jest niższy od Bartka. Kto jest

```


najniższy?

Rozważmy to krok po kroku:

Pytanie: Co wiem o Jacku i Tomku?
Odpowiedź: Jacek jest wyższy od Tomka
Pytanie: Co wiem o Adamie i Jacku?
Odpowiedź: Adam jest wyższy od Jacka
Pytanie: Co wiem o Adamie i Bartku?
Odpowiedź: Bartek jest wyższy od Adama
Pytanie: Kto jest najniższy?
Odpowiedź: Tomek

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s

"""

    self.run_technique(learning_prompt)

def meta_technique(self):
    """ Meta technique for HeightProblem """

    learning_prompt = """
```

Jacek jest wyższy od Tomka, ale niższy od Adama. Adam jest niższy od Bartka. Kto jest najniższy?

Aby odpowiedzieć na to pytanie, musimy najpierw zrozumieć kolejność:

1. Jacek > Tomek
2. Adam > Jacek
3. Bartek > Adam

Teraz możemy ustalić, że:

Bartek > Adam > Jacek > Tomek

Więc najniższy jest Tomek.

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s

"""

    self.run_technique(learning_prompt)

def chain_of_thought_technique(self):
    """ Chain-of-thought technique for HeightProblem """

    learning_prompt = """
```

Jacek jest wyższy od Tomka, ale niższy od Adama. Adam jest niższy od Bartka. Kto jest najniższy?

Zacznijmy od porządku:

- Jacek jest wyższy od Tomka, więc Jacek > Tomek
- Adam jest wyższy od Jacka, więc Adam > Jacek > Tomek
- Bartek jest wyższy od Adama, więc Bartek > Adam > Jacek > Tomek

Więc najniższy jest Tomek.

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s

"""

    self.run_technique(learning_prompt)

def re_act_technique(self):
    """ ReAct technique for HeightProblem """

    learning_prompt = """
```

Jacek jest wyższy od Tomka, ale niższy od Adama. Adam jest niższy od Bartka. Kto jest

```

najniższy?

Analizuję fakty:
- Jacek > Tomek
- Adam > Jacek
- Bartek > Adam

Na podstawie tych informacji, ustalam porządek:
    Bartek > Adam > Jacek > Tomek

Więc najniższy jest Tomek.

Wykorzystaj to podejście do rozwiązania tego problemu:
%s

    """

    self.run_technique(learning_prompt)

if __name__ == "__main__":
    height_problem = HeightProblem()
    # height_problem.least_to_most_technique()
    # height_problem.self_asking_technique()
    # height_problem.meta_technique()
    # height_problem.chain_of_thought_technique()
    # height_problem.re_act_technique()

```

Klasa dla problemów czasu

```

from ProblemsBase import ProblemsBase

class TimeProblem(ProblemsBase):
    """ TimeProblem class for time-related problems to be solved by different prompt
    engineering techniques """

    def __init__(self):
        super(TimeProblem, self).__init__("time_problems.json")

    def least_to_most_technique(self):
        """ Least-to-most technique for TimeProblem """

        learning_prompt = """
Jeśli dziś jest środa, to który dzień tygodnia będzie za 100 dni?

Rozkładamy problem na małe problemy:
Pytanie: Ile dni jest w tygodniu?
Odpowiedź: 7 dni
Pytanie: Ile pełnych tygodni mieści się w 100 dniach?
Odpowiedź: 14 tygodni (ponieważ 100 podzielone przez 7 to 14 z resztą 2)
Pytanie: Jaki jest dzień tygodnia po 100 dniach, jeśli reszta z dzielenia to 2?
Odpowiedź: Piątek (ponieważ dwa dni po środzie to piątek)

Poprawna odpowiedź to: Piątek

Wykorzystaj to podejście do rozwiązania tego problemu:
%s

    """

    self.run_technique(learning_prompt)

    def self_asking_technique(self):
        """ Self-asking technique for TimeProblem """

        learning_prompt = """

```

Jeśli dziś jest środa, to który dzień tygodnia będzie za 100 dni?

Rozważmy to krok po kroku:

Pytanie: Ile dni jest w tygodniu?

Odpowiedź: 7 dni

Pytanie: Jak obliczyć dzień tygodnia po 100 dniach?

Odpowiedź: Podzielić 100 przez 7, co daje 14 tygodni i resztę 2 dni

Pytanie: Jeśli dziś jest środa, jaki dzień będzie za 2 dni?

Odpowiedź: Piątek

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
```

```
"""
```

```
self.run_technique(learning_prompt)
```

```
def meta_technique(self):
```

```
    """ Meta technique for TimeProblem """
```

```
    learning_prompt = """
```

Jeśli dziś jest środa, to który dzień tygodnia będzie za 100 dni?

Aby odpowiedzieć na to pytanie, musimy najpierw zrozumieć kolejność:

1. Tydzień ma 7 dni

2. 100 dni podzielone przez 7 dni to 14 tygodni z resztą 2 dni

Teraz możemy ustalić, że:

2 dni po środzie to piątek

Więc odpowiedź to piątek.

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
```

```
"""
```

```
self.run_technique(learning_prompt)
```

```
def chain_of_thought_technique(self):
```

```
    """ Chain-of-thought technique for TimeProblem """
```

```
    learning_prompt = """
```

Jeśli dziś jest środa, to który dzień tygodnia będzie za 100 dni?

Zacznijmy od porządku:

- Tydzień ma 7 dni

- 100 dni to 14 tygodni z resztą 2 dni

- Jeśli dziś jest środa, to za 2 dni będzie piątek

Więc odpowiedź to piątek.

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
```

```
"""
```

```
self.run_technique(learning_prompt)
```

```
def re_act_technique(self):
```

```
    """ ReAct technique for TimeProblem """
```

```
    learning_prompt = """
```

Jeśli dziś jest środa, to który dzień tygodnia będzie za 100 dni?

Analizuję fakty:

- Tydzień ma 7 dni

- 100 dni to 14 tygodni z resztą 2 dni

Na podstawie tych informacji, ustalam porządek:
- Jeśli dziś jest środa, to za 2 dni będzie piątek

Więc odpowiedź to piątek.

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
    """

    self.run_technique(learning_prompt)

if __name__ == "__main__":
    time_problem = TimeProblem()
    # time_problem.least_to_most_technique()
    # time_problem.self_asking_technique()
    # time_problem.meta_technique()
    # time_problem.chain_of_thought_technique()
    # time_problem.react_technique()
```

Klasa dla problemów liter

```
from ProblemsBase import ProblemsBase

class LettersProblem(ProblemsBase):
    """ LettersProblem class for letter-related problems to be solved by different
    prompt engineering techniques """

    def __init__(self):
        super(LettersProblem, self).__init__("letters_problems.json")

    def least_to_most_technique(self):
        """ Least-to-most technique for LettersProblem """

        learning_prompt = ""
```

Ułóż litery „EALNG” w kolejności alfabetycznej.

Rozkładamy problem na małe problemy:

```
Pytanie: Jakie litery mamy?
Odpowiedź: E, A, L, N, G
Pytanie: Jaka jest pierwsza litera w kolejności alfabetycznej?
Odpowiedź: A
Pytanie: Jaka jest kolejna litera po A?
Odpowiedź: E
Pytanie: Jaka jest kolejna litera po E?
Odpowiedź: G
Pytanie: Jaka jest kolejna litera po G?
Odpowiedź: L
Pytanie: Jaka jest kolejna litera po L?
Odpowiedź: N
```

Poprawna odpowiedź to: A, E, G, L, N

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
    """

    self.run_technique(learning_prompt)

    def self_asking_technique(self):
        """ Self-asking technique for LettersProblem """

        learning_prompt = ""
Ułóż litery „EALNG” w kolejności alfabetycznej.
```

Rozważmy to krok po kroku:

Pytanie: Jakie litery mamy?

Odpowiedź: E, A, L, N, G

Pytanie: Jak obliczyć kolejność alfabetyczną liter?

Odpowiedź: Porównaj każdą literę, zaczynając od A

Pytanie: Jakie litery mamy po A?

Odpowiedź: E

Pytanie: Jakie litery mamy po E?

Odpowiedź: G

Pytanie: Jakie litery mamy po G?

Odpowiedź: L

Pytanie: Jakie litery mamy po L?

Odpowiedź: N

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
```

```
"""
```

```
self.run_technique(learning_prompt)
```

```
def meta_technique(self):
```

```
    """ Meta technique for LettersProblem """
```

```
    learning_prompt = """
```

Ułóż litery „EALNG” w kolejności alfabetycznej.

Aby odpowiedzieć na to pytanie, musimy najpierw zrozumieć kolejność alfabetyczną:

1. Rozpoznajemy litery: E, A, L, N, G

2. Porządkujemy je alfabetycznie: A, E, G, L, N

Poprawna odpowiedź to: A, E, G, L, N

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
```

```
"""
```

```
self.run_technique(learning_prompt)
```

```
def chain_of_thought_technique(self):
```

```
    """ Chain-of-thought technique for LettersProblem """
```

```
    learning_prompt = """
```

Ułóż litery „EALNG” w kolejności alfabetycznej.

Zacznijmy od porządku:

- Mamy litery: E, A, L, N, G

- W kolejności alfabetycznej są: A, E, G, L, N

Więc odpowiedź to: A, E, G, L, N

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
```

```
"""
```

```
self.run_technique(learning_prompt)
```

```
def re_act_technique(self):
```

```
    """ ReAct technique for LettersProblem """
```

```
    learning_prompt = """
```

Ułóż litery „EALNG” w kolejności alfabetycznej.

Analizuję fakty:

- Mamy litery: E, A, L, N, G

- Uporządkujemy je alfabetycznie: A, E, G, L, N

Na podstawie tych informacji ustalamy, że poprawna kolejność to: A, E, G, L, N

Wykorzystaj to podejście do rozwiązania tego problemu:

```
%s
    """

    self.run_technique(learning_prompt)

if __name__ == "__main__":
    letters_problem = LettersProblem()
    letters_problem.least_to_most_technique()
    letters_problem.self_asking_technique()
    letters_problem.meta_technique()
    letters_problem.chain_of_thought_technique()
    letters_problem.re_act_technique()
```

Uruchamianie sprawdzania

Każde uruchomienie metody to uruchomienie danego problemu dla danej techniki. Każdy problem będzie poprzedzony rozwiązaniem jego w ramach jakiejś techniki tak, żeby pokazać chatowi w jaki sposób ma podejść do problemu. Są zapisane w plikach tekstowych, swoją strukturą wyglądają tak samo jak ten przedstawiony na początku dla różnych problemów.

Wyniki w poprawności wyrażone są poniżej w postaci tabeli:

Technika	Problem wzrostu	Problem czasu	Problem liter	Poprawne	Wynik
Liczba problemów	11	11	11	33	100%
Least-to-most	9	8	11	28	85%
Self-asking	7	7	10	24	73%
Meta	7	9	11	27	82%
Chain-of-thought	8	6	11	25	76%
ReAct	9	9	11	29	88%

Analiza wyników

Jak widać na tabeli, model językowi zareagował z różną skutecznością w zależności od rodzaju problemu jaki dostał i techniki, która służyła jako wskazówkę jak do niego podejść.

Najgorszym podejściem okazało się „Self-asking”, wymagało to od modelu stworzenia własnych pytań, przez co poza problemem z podaniem dobrego wyniku pojawił się również problem z opracowaniem poprawnych odpowiedzi.

Jako najlepszą technikę wyniki wskazują na „ReAct”. Dzięki stworzeniu schematu dedukcji na podstawie analizy faktów i określania działania potrafił on osiągnąć dużą skuteczność w udzielaniu poprawnych odpowiedzi.

Analizując wszystkie techniki, każda z nich dała wynik ponad 70% co oznacza, że żadnej nie warto wykluczać jako nie działającą. Każda z nich będzie działać inaczej dla innych problemów.

Podsumowanie

Udało się skutecznie stworzyć wszystkie mechanizmy sprawdzające skuteczność analizowanych technik inżynierii odpowiedzi. Dokonana analiza pokazuje różnice w ich skuteczności w zależności od problemu.

Jak pokazuje analiza wyników, techniki mają różną skuteczność, co wskazuje na to, że mają one różne zastosowanie. Warto planując złożone zapytania do modelu językowego zastanowić się nad dobraniem odpowiedniej techniki do problemu, przed jakim stoimy.

Widać również różnicę w skuteczności starszego modelu językowego „gpt-3.5-turbo” w porównaniu do niedawno wydanego „gpt-4o”, który po dodatkowych testach pokazuje prawie pełną skuteczność w zadaniach o poziomie takim, jakie miały te wykorzystywane w projekcie.

Końcowo, wyniki pokazują, że dziedzina sztucznej inteligencji nadal ma przestrzeń do rozwoju, oraz techniki inżynierii odpowiedzi nadal muszą się rozwijać odpowiadając na wyzwania poprawnej i trafnej komunikacji z modelami językowymi.