ДОДАТОК А

Лістинг програмного коду

graph_builder.py

```
import spacy
import numpy as np
import pandas as pd
import os
import re
import pandas as pd
import bs4
import requests
import spacy
from spacy import displacy

from spacy.matcher import Matcher
from spacy.tokens import Span

import networkx as nx

import matplotlib.pyplot as plt
from tqdm import tqdm
import coreferee
from pathlib import Path
import pandas as pd
from spacy.matcher import Matcher

nlp = spacy.load('en_core_web_sm')
nlp.add_pipe("coreferee")

PRONOUNS = {
    "i","you","he","she","it","we","they", "which",

"me","him","her","us","them","this","that","these","those",
```

```
    "therefore"
}

dir_path = Path("./texts")

# Replace pronouns using token-level replacement
def resolve_pronouns(doc):
    tokens = [tok.text for tok in doc]
    if doc._.coref_chains:
        for chain in doc._.coref_chains:
            main = chain.main
            for mention in chain:
                if mention != main:
                    for i in range(mention.start,
mention.end):
                        tokens[i] = main.text
    return " ".join(tokens)

def is_valid_entity(ent):
    """Check if an entity is non-empty and does not
contain any pronoun."""
    if not ent or not ent.strip():
        return False
    ent_lower = ent.strip().lower()

    return not any(pronoun in ent_lower.split() for
pronoun in PRONOUNS)

def load_text():
    all_sentences = []

    for file_path in dir_path.glob("*.txt"):
        with open(file_path, "r", encoding="utf-8") as
f:
            text = f.read()

```

```
        doc = nlp(text)

        # Coreference chains
        for chain in doc._.coref_chains:
            print(chain)  # inspect chains

        # Split into sentences
        for sent in doc.sents:
            s = sent.text.strip().lower()
            if s:
                all_sentences.append(s)

        candidate_sentences                          =
pd.DataFrame(all_sentences, columns=["sentence"])

        return candidate_sentences

def get_entities(sent):
    ## chunk 1
    ent1 = ""
    ent2 = ""

    prv_tok_dep = ""   # dependency tag of previous
token in the sentence
    prv_tok_text = ""  # previous token in the sentence

    prefix = ""
    modifier = ""


#############################################################

    for tok in nlp(sent):
        ## chunk 2
        # if token is a punctuation mark then move on
to the next token
```

```
        if tok.dep_ != "punct":
            # check: token is a compound word or not
            if tok.dep_ == "compound":
                prefix = tok.text
                # if the previous word was also a
'compound' then add the current word to it
                if prv_tok_dep == "compound":
                    prefix = prv_tok_text + " " +
tok.text

            # check: token is a modifier or not
            if tok.dep_.endswith("mod") == True:
                modifier = tok.text
                # if the previous word was also a
'compound' then add the current word to it
                if prv_tok_dep == "compound":
                    modifier = prv_tok_text + " " +
tok.text

            ## chunk 3
            if tok.dep_.find("subj") == True:
                ent1 = modifier + " " + prefix + " "
+ tok.text
                prefix = ""
                modifier = ""
                prv_tok_dep = ""
                prv_tok_text = ""

                ## chunk 4
            if tok.dep_.find("obj") == True:
                ent2 = modifier + " " + prefix + " "
+ tok.text

            ## chunk 5
            # update variables
            prv_tok_dep = tok.dep_
```

```
            prv_tok_text = tok.text

##############################################################

    return [ent1.strip(), ent2.strip()]

def get_entity_pairs(candidate_sentences):
    entity_pairs = []

    for i in tqdm(candidate_sentences["sentence"]):
        entity_pairs.append(get_entities(i))

    return entity_pairs

def get_relation(sent):
    doc = nlp(sent)

    matcher = Matcher(nlp.vocab)

    # Define the pattern
    pattern = [
        {'DEP':'ROOT'},
        {'DEP':'prep','OP':"?"},
        {'DEP':'agent','OP':"?"},
        {'POS':'ADJ','OP':"?"}
    ]

    matcher.add("matching_1", [pattern])

    matches = matcher(doc)

    if not matches:
        return None

    k = len(matches) - 1
    span = doc[matches[k][1]:matches[k][2]]
```

```
    return span.text

def get_relations(candidate_sentences):
    return [get_relation(i) for i in
tqdm(candidate_sentences['sentence'])]

def build_kg_df(entity_pairs, relations):
    # Extract subject and object
    source = [i[0] for i in entity_pairs]
    target = [i[1] for i in entity_pairs]

    # Filter out invalid pairs
    valid_pairs = [(s, t, r) for s, t, r in zip(source,
target, relations) if is_valid_entity(s) and is_valid_entity(t)]

    # Unpack filtered lists
    source_filtered = [s for s, t, r in valid_pairs]
    target_filtered = [t for s, t, r in valid_pairs]
    relations_filtered = [r for s, t, r in valid_pairs]

    # Build the knowledge graph DataFrame
    kg_df = pd.DataFrame({
        'source': source_filtered,
        'target': target_filtered,
        'edge': relations_filtered
    })

    return kg_df

def build_knowledge_graph():
    candidate_sentences = load_text()
    entity_pairs                                    =
get_entity_pairs(candidate_sentences)
    relations = get_relations(candidate_sentences)
    kg_df = build_kg_df(entity_pairs, relations)
```

```
    [redacted]
    [redacted]    G=nx.from_pandas_edgelist(kg_df,        "source",
"target",
    [redacted]                              edge_attr=True,
create_using=nx.MultiDiGraph())
    [redacted]
        return G
```

ДОДАТОК Б

Промпт для генерації моделлю тестових завдань

```
You are a system designed to generate educational multiple-choice test questions.

Generate exactly {{questions_num}} multiple-choice questions strictly based on the topic and the provided context.

Do not invent facts; rely only on the given information.

User topic:
{{topic}}

Relevant context:
{{rag_context}}

Requirements for the test questions:
- Each question must follow the multiple-choice format.
- Use only the knowledge that appears in the topic and the provided context.
- Each question must include one correct answer and several plausible distractors.
- Avoid duplicate or overly similar questions.
- Do not provide explanations, comments, or additional text outside the required format.

STRICT OUTPUT FORMAT:
Return the result as a valid JSON array, following this structure:
[
  {
    "question": "text of the question",
    "answers": ["option1", "option2", "option3", "option4"],
```

```
    "correct_answer": "one of the answer options above"
  },
  ...
]

Do not output anything outside the JSON array.
```

ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	74 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Жеребний В.В.			Розробка генеративної моделі створення навчального контенту на основі великих мовних моделей	Шифр групи	Код напр./спец.
Перевірив		Магдаліна І.В.				ДСм-24-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Чала Л.Е.					