#train.py

import random

import nltk

from nltk.corpus import wordnet

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk import pos\_tag

# Download NLTK data (if not already downloaded)

nltk.download('punkt', quiet=True)

nltk.download('wordnet', quiet=True)

nltk.download('averaged\_perceptron\_tagger', quiet=True)

nltk.download('omw-1.4', quiet=True)  # Open Multilingual WordNet

def get\_wordnet\_pos(tag):

    """Map NLTK POS tag to WordNet POS tag"""

    if tag.startswith('J'):

        return wordnet.ADJ

    elif tag.startswith('V'):

        return wordnet.VERB

    elif tag.startswith('N'):

        return wordnet.NOUN

    elif tag.startswith('R'):

        return wordnet.ADV

    else:

        return None

def replace\_synonyms(text, synonym\_chance=0.7, max\_candidates=5):

    """

    Replace words in the text with their synonyms, ensuring grammatical correctness.

    Args:

        text (str): The input text.

        synonym\_chance (float): Probability of replacing a word with synonym.

        max\_candidates (int): Maximum number of candidate synonyms to consider.

    Returns:

        str: The paraphrased text.

    """

    words = word\_tokenize(text)

    pos\_tags = pos\_tag(words)

    paraphrased\_words = []

    for i, (word, tag) in enumerate(pos\_tags):

        # Skip short words, punctuation, and common words

        if len(word) <= 3 or not word.isalpha() or word.lower() in {'the', 'and', 'to', 'of', 'a', 'in', 'is', 'that', 'for', 'it', 'as', 'be', 'on', 'with'}:

            paraphrased\_words.append(word)

            continue

        # Only attempt synonym replacement with probability synonym\_chance

        if random.random() > synonym\_chance:

            paraphrased\_words.append(word)

            continue

        wordnet\_pos = get\_wordnet\_pos(tag)

        if not wordnet\_pos:

            paraphrased\_words.append(word)

            continue

        # Get all synsets for the word with the correct POS

        synsets = wordnet.synsets(word, pos=wordnet\_pos)

        if not synsets:

            paraphrased\_words.append(word)

            continue

        # Get unique lemmas from all synsets

        lemmas = set()

        for synset in synsets:

            for lemma in synset.lemmas():

                if lemma.name().lower() != word.lower() and "\_" not in lemma.name():

                    lemmas.add(lemma.name().lower())

        # Convert to list and limit candidates

        lemmas = list(lemmas)[:max\_candidates]

        if lemmas:

            # Choose a random synonym

            synonym = random.choice(lemmas)

            # Match original capitalization

            if word[0].isupper():

                synonym = synonym.capitalize()

            paraphrased\_words.append(synonym)

        else:

            paraphrased\_words.append(word)

    # Handle spacing around punctuation

    result = " ".join(paraphrased\_words)

    for punct in [',', '.', '!', '?', ':', ';']:

        result = result.replace(f" {punct}", punct)

    return result

def reorder\_phrases(sentence):

    """

    Attempt to reorder phrases in a sentence while maintaining meaning.

    Only applies to certain sentence structures.

    """

    # Simple implementation - for more complex reordering, you'd need a parser

    if ", " in sentence and random.random() > 0.5:

        parts = sentence.split(", ", 1)

        if len(parts) == 2 and not any(w in parts[0].lower() for w in ["if", "when", "while", "because"]):

            # Ensure the second part can stand as its own clause

            if any(parts[1].startswith(w) for w in ["and ", "but ", "however "]):

                return sentence  # Don't reorder in this case

            return parts[1] + ", " + parts[0]

    return sentence

def generate\_paraphrases(text, num\_paraphrases=3, variation\_level="medium"):

    """

    Generate paraphrases for the given text with different variation levels.

    Args:

        text (str): The input text.

        num\_paraphrases (int): Number of paraphrases to generate.

        variation\_level (str): Level of variation - "low", "medium", or "high".

    Returns:

        list: A list of paraphrased sentences.

    """

    # Set parameters based on variation level

    if variation\_level == "low":

        synonym\_chance = 0.3

        reorder\_chance = 0.1

    elif variation\_level == "medium":

        synonym\_chance = 0.5

        reorder\_chance = 0.3

    else:  # high

        synonym\_chance = 0.7

        reorder\_chance = 0.5

    sentences = sent\_tokenize(text)

    paraphrases = []

    for \_ in range(num\_paraphrases):

        paraphrased\_sentences = []

        for sentence in sentences:

            # Replace synonyms

            paraphrased = replace\_synonyms(sentence, synonym\_chance)

            # Potentially reorder phrases

            if random.random() < reorder\_chance:

                paraphrased = reorder\_phrases(paraphrased)

            paraphrased\_sentences.append(paraphrased)

        paraphrased\_text = " ".join(paraphrased\_sentences)

        paraphrases.append(paraphrased\_text)

    return paraphrases

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    text = "Artificial intelligence (AI) is transforming the world in unprecedented ways. From healthcare to finance, AI is being used to solve complex problems and improve efficiency."

    print("Original text:")

    print(text)

    print("\nParaphrases:")

    for level in ["low", "medium", "high"]:

        print(f"\n{level.capitalize()} variation level:")

        paraphrases = generate\_paraphrases(text, num\_paraphrases=2, variation\_level=level)

        for i, paraphrase in enumerate(paraphrases):

            print(f"{i+1}. {paraphrase}")

#app.py

import streamlit as st

from train import generate\_paraphrases

# Streamlit app setup

st.set\_page\_config(page\_title="Enhanced Paraphrase Generator")

# App title and description

st.title("📝 Paraphrase Generator")

st.markdown("""

This application generates paraphrases of your text

""")

# Input section

st.subheader("Input Text")

user\_input = st.text\_area("Enter your text to paraphrase (up to 500 words):", height=150)

# Configuration options

col1, col2 = st.columns(2)

with col1:

    num\_paraphrases = st.slider("Number of paraphrases:", min\_value=1, max\_value=5, value=3)

with col2:

    variation\_level = st.select\_slider(

        "Variation level:",

        options=["low", "medium", "high"],

        value="medium",

        help="Low: Subtle changes. Medium: Moderate substitutions. High: Maximum variation."

    )

# Process button

if st.button("Generate Paraphrases", type="primary"):

    if not user\_input:

        st.error("Please enter some text to paraphrase.")

    else:

        with st.spinner("Generating paraphrases..."):

            try:

                paraphrases = generate\_paraphrases(

                    user\_input,

                    num\_paraphrases=num\_paraphrases,

                    variation\_level=variation\_level

                )

                # Display results

                st.subheader("Paraphrased Results")

                # Original text for comparison

                with st.expander("Original Text", expanded=True):

                    st.write(user\_input)

                # Display each paraphrase

                for i, paraphrase in enumerate(paraphrases):

                    with st.expander(f"Paraphrase {i+1}", expanded=True):

                        st.write(paraphrase)

                # Download options

                st.download\_button(

                    label="Download Results as Text",

                    data="\n\nORIGINAL:\n" + user\_input + "\n\n" +

                         "\n\n".join([f"PARAPHRASE {i+1}:\n{p}" for i, p in enumerate(paraphrases)]),

                    file\_name="paraphrases.txt",

                    mime="text/plain"

                )

            except Exception as e:

                st.error(f"An error occurred: {str(e)}")

# Add information about the method

with st.expander("How it works"):

    st.markdown("""

    ### Rule-based Paraphrasing Technique

    This enhanced paraphraser uses several techniques:

    1. \*\*Intelligent Synonym Replacement\*\*: Words are replaced with synonyms based on their part-of-speech, with care to maintain grammatical correctness.

    2. \*\*Selective Replacement\*\*: Common words, short words, and function words are preserved to maintain readability.

    3. \*\*Phrase Reordering\*\*: When possible, clauses in complex sentences are rearranged while preserving meaning.

    4. \*\*Variable Randomness\*\*: The variation level controls how aggressive the paraphrasing should be.

    5. \*\*Formatting Preservation\*\*: Proper capitalization and punctuation are maintained.

    Unlike deep learning approaches, this method is lightweight, requires no training data, and runs efficiently on any machine.

    """)

st.markdown("---")

st.caption("Done By a TEAM")