* 6.Passive voice program

import spacy

import nltk

from nltk.corpus import wordnet

from nltk.stem import WordNetLemmatizer

# Download necessary resources

nltk.download('wordnet')

nltk.download('omw-1.4')

# Load Spacy English model

nlp = spacy.load("en\_core\_web\_sm")

# Dictionary for handling irregular verbs manually

irregular\_verbs = {

    "eat": "eaten", "write": "written", "break": "broken", "see": "seen", "take": "taken",

    "give": "given", "drive": "driven", "speak": "spoken", "choose": "chosen", "forget": "forgotten",

    "steal": "stolen", "freeze": "frozen", "ride": "ridden", "fall": "fallen", "do": "done",

    "go": "gone", "be": "been"

}

def get\_past\_participle(verb):

    """Convert a verb to its past participle form."""

    lemmatizer = WordNetLemmatizer()

    lemma = lemmatizer.lemmatize(verb, "v")  # Get base form of the verb

    # Check for irregular verbs first

    if lemma in irregular\_verbs:

        return irregular\_verbs[lemma]

    # Attempt to get past participle using WordNet

    synsets = wordnet.synsets(lemma, pos=wordnet.VERB)

    for syn in synsets:

        for lemma in syn.lemmas():

            for related\_form in lemma.derivationally\_related\_forms():

                if related\_form.name().endswith("en"):  # Check for common past participle suffix

                    return related\_form.name()

    return lemma + "ed"  # Default to adding "ed" for regular verbs

def convert\_to\_passive(sentence):

    """Convert an active voice sentence to passive voice."""

    doc = nlp(sentence)

    subject = None

    verb = None

    obj = None

    for token in doc:

        if "subj" in token.dep\_:

            subject = token

        if token.pos\_ == "VERB":

            verb = token

        if "obj" in token.dep\_:

            obj = token

    if subject and verb and obj:

        past\_participle = get\_past\_participle(verb.text)

        passive\_sentence = f"{obj.text} was {past\_participle} by {subject.text}."

        return passive\_sentence.capitalize()

    else:

        return "Could not convert to passive voice."

# Take user input

active\_sentence = input("Enter an active voice sentence: ")

passive\_sentence = convert\_to\_passive(active\_sentence)

# Display output

print("\nActive Voice:  ", active\_sentence)

print("Passive Voice: ", passive\_sentence)

* 7.program Ngram model

import nltk

from nltk.util import ngrams

from nltk.corpus import words

from difflib import get\_close\_matches

# Download English words corpus (if not already downloaded)

nltk.download('words')

# Load dictionary of English words

word\_list = set(words.words())

def generate\_ngrams(word, n=2):

    """Generate n-grams for a given word"""

    return [''.join(gram) for gram in ngrams(word, n)]

def suggest\_correction(misspelled\_word, n=2):

    """Suggest correct words based on n-gram similarity"""

    misspelled\_ngrams = set(generate\_ngrams(misspelled\_word, n))

    suggestions = {}

    for correct\_word in word\_list:

        correct\_ngrams = set(generate\_ngrams(correct\_word, n))

        similarity = len(misspelled\_ngrams & correct\_ngrams) / max(len(misspelled\_ngrams), len(correct\_ngrams))

        suggestions[correct\_word] = similarity

    # Get top 3 suggestions

    sorted\_suggestions = sorted(suggestions, key=suggestions.get, reverse=True)[:3]

    return sorted\_suggestions

# Example usage

misspelled = input("Enter a misspelled word: ")

corrections = suggest\_correction(misspelled, n=3)

if corrections:

    print(f"Did you mean: {', '.join(corrections)}?")

else:

    print("No suggestions found.")

* 8.Program HMM for posttag

import nltk

from nltk.corpus import treebank

from nltk.tag import hmm

from sklearn.model\_selection import train\_test\_split

# Download necessary data

nltk.download('treebank')

nltk.download('universal\_tagset')

# Load the dataset

data = treebank.tagged\_sents(tagset='universal')

# Split into training and test sets

train\_data, test\_data = train\_test\_split(data, test\_size=0.2, random\_state=42)

# Train the HMM tagger

trainer = hmm.HiddenMarkovModelTrainer()

tag\_model = trainer.train(train\_data)

# Predict and evaluate

def evaluate\_hmm(tagger, test\_data):

    accuracy = tagger.evaluate(test\_data)

    return accuracy

# Example usage

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

    accuracy = evaluate\_hmm(tag\_model, test\_data)

    print(f"HMM POS Tagger Accuracy: {accuracy:.4f}")

    # Test on a sample sentence

    sentence = ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]

    predicted\_tags = tag\_model.tag(sentence)

    print("POS Tags:", predicted\_tags)

* 9.PROGRAM TRANSLATE

from googletrans import Translator

def translate\_text(text, target\_lang):

    translator = Translator()

    try:

        translated = translator.translate(text, dest=target\_lang)

        return translated.text

    except Exception as e:

        return f"Error: {e}"

# List of common language codes

language\_codes = {

    "tamil": "ta",

    "hindi": "hi",

    "malayalam": "ml",

    "french": "fr",

    "spanish": "es",

    "german": "de",

    "chinese": "zh-cn",

    "japanese": "ja"

}

while True:

    text = input("Enter English text (or type 'exit' to quit): ")

    if text.lower() == "exit":

        break

    print("Available languages:", ", ".join(language\_codes.keys()))

    lang = input("Enter target language: ").lower()

    if lang in language\_codes:

        translated\_text = translate\_text(text, language\_codes[lang])

        print(f"Translated ({lang}): {translated\_text}\n")

    else:

        print("❌ Invalid language. Please try again.\n")

* 10.PROGRAM CHATBOT

import nltk

from nltk.chat.util import Chat, reflections

# Define predefined responses using pattern-response pairs

pairs = [

    (r"hi|hello|hey", ["Hello!", "Hey there!", "Hi! How can I assist you?"]),

    (r"how are you?", ["I'm just a bot, but I'm doing great!", "I'm fine, thanks for asking!"]),

    (r"what is your name?", ["I'm a chatbot!", "You can call me ChatBot."]),

    (r"who created you?", ["I was created by a programmer!", "I'm a result of some cool coding skills."]),

    (r"what can you do?", ["I can chat with you, answer simple questions, and keep you entertained!"]),

    (r"bye|goodbye", ["Goodbye!", "See you soon!", "Bye! Have a great day!"]),

    (r"(.\*)", ["I'm not sure how to respond to that.", "Can you ask something else?", "Interesting! Tell me more."])

]

# Create a chatbot instance

chatbot = Chat(pairs, reflections)

# Start chatting with the user

def start\_chat():

    print("Hello! I'm your chatbot. Type 'bye' to exit.")

    while True:

        user\_input = input("You: ").lower()

        if user\_input in ["bye", "exit", "quit"]:

            print("ChatBot: Goodbye! Have a great day!")

            break

        response = chatbot.respond(user\_input)

        print("ChatBot:", response)

# Run the chatbot

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

    start\_chat()

* 11.PROGRAM UNIFICATION CONSTRAINT

# Import necessary libraries

import nltk

from nltk import CFG

from nltk.parse import ChartParser

from nltk.tokenize import word\_tokenize

# Define the grammar for subject-verb agreement

grammar\_string = """

    S -> NP VP

    VP -> V NP | V

    NP -> Det N | Det N PP | N

    PP -> P NP

    Det -> 'a' | 'the'

    N -> 'man' | 'dog' | 'cat' | 'telescope' | 'park'

    V -> 'saw' | 'ate' | 'walked'

    P -> 'in' | 'on' | 'by' | 'with'

"""

# Define the grammar checker function

def grammar\_checker(sentence):

    # Load the grammar

    grammar = CFG.fromstring(grammar\_string)

    parser = ChartParser(grammar)

    # Tokenize the sentence

    tokens = word\_tokenize(sentence)

    # Parse the sentence and generate parse trees

    try:

        parse\_trees = list(parser.parse(tokens))

        if parse\_trees:

            print("Grammar is correct\n")

            for tree in parse\_trees:

                tree.pretty\_print()  # Display parse tree

            return "Parsing successful!"

    except Exception as e:

        return "Grammar is incorrect: " + str(e)

# Example usage

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

    input\_sentence = input("Enter a sentence: ")

    result = grammar\_checker(input\_sentence)

    print(result)

* 12.PROGRAM WSD

import nltk

import spacy

from nltk.corpus import wordnet

from nltk.wsd import lesk

# Download necessary resources

nltk.download('wordnet')

nltk.download('omw-1.4')

nltk.download('punkt')

# Load SpaCy model

nlp = spacy.load("en\_core\_web\_sm")

def lesk\_wsd(word, sentence):

    """Apply Lesk's algorithm to find the best sense of a word."""

    return lesk(nltk.word\_tokenize(sentence), word)

def syntax\_analysis(sentence):

    """Perform syntax-driven semantic analysis."""

    doc = nlp(sentence)

    relations = {}

    for token in doc:

        if token.dep\_ in ["nsubj", "dobj"]:  # Subject & Object relations

            relations[token.text] = (token.dep\_, token.head.text)  # (relation, governing verb)

    return relations

def hybrid\_wsd(sentence, target\_word):

    """Hybrid WSD combining Lesk & Syntax Analysis."""

    # Apply Dictionary-Based WSD (Lesk Algorithm)

    lesk\_sense = lesk\_wsd(target\_word, sentence)

    # Apply Syntax-Driven Semantic Analysis

    syntax\_relations = syntax\_analysis(sentence)

    # If the target word is found in a grammatical role, refine selection

    if target\_word in syntax\_relations:

        role, governing\_verb = syntax\_relations[target\_word]

        # Check if the governing verb aligns with the sense’s examples

        best\_sense = None

        max\_overlap = 0

        for synset in wordnet.synsets(target\_word):

            example\_overlap = sum(1 for example in synset.examples() if governing\_verb in example)

            if example\_overlap > max\_overlap:

                best\_sense = synset

                max\_overlap = example\_overlap

        if best\_sense:

            return best\_sense.definition()

    # If no better refinement is found, return the Lesk-based sense

    return lesk\_sense.definition() if lesk\_sense else "No sense found."

# Take runtime input

sentence = input("Enter a sentence: ")

target\_word = input("Enter the word to disambiguate: ")

# Perform Hybrid WSD

sense = hybrid\_wsd(sentence, target\_word)

# Display result

print(f"\nBest sense for '{target\_word}': {sense}")