INDEX

SI.NO	DATE	EXPERIMENTS	PAGE NO.	SIGN
1		Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmeticoperations and analyze results.		
2		Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1. Select 10 words from aspecific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships.Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar wordsfor a given input.		
3		Train a custom Word2Vec model on a small dataset. Train embeddings on a domain-specific corpus (e.g., legal,medical) and analyze how embeddings capture domain-specific semantics.		
4		Use word embeddings to improve prompts for Generative AI model. Retrieve similar words using wordembeddings. Use the similar words to enrich a GenAI prompt. Use the AI model to generate responses for theoriginal and enriched prompts. Compare the outputs in terms of detail and relevance.		
5		Use word embeddings to create meaningful sentences for creative tasks. Retrieve similar words for a seed word.Create a sentence or story using these words as a starting point. Write a program that: Takes a seed word. Generatessimilarwords.		

		Constructs a short paragraph using these words.		
6		**Use a pre-trained Hugging Face model to analyze sentiment in text. Assume a real-world application, Load thesentiment analysis pipeline. Analyze the sentiment by giving sentences to input.**		
7		**Summarize long texts using a pre-trained summarization model using Hugging face model. Load thesummarization pipeline. Take a passage as input and obtain the summarized text.**		
8		**Install langchain, cohere (for key), langchain-community. Get the apikey(By logging into Cohere and obtainingthe cohere key). Load a text document from your google drive . Create a prompt template to display the output ina particular manner.**		
9		**Take the Institution name as input. Use Pydantic to define the schema for the desired output and create a custom output parser. Invoke the Chain and Fetch Results. Extract the below Institution related details from Wikipedia: The founder of the Institution. When it was founded. The current branches in the institution . How manyemployees are working in it. A brief 4-line summary of the institution.**		
10		**Build a chatbot for the Indian Penal Code. We'll start by downloading the official Indian Penal Code document,and then we'll create a chatbot that can interact with it. Users will be able to ask questions about the Indian PenalCode and have a conversation with it.**		

1.Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmeticoperations and analyze results.

Source Code:

```
import gensim.downloader as api
import numpy as np
# Load pre-trained GloVe embeddings (100d)
word_vectors = api.load('glove-wiki-gigaword-100')  # ~130MB
print(f"Vocabulary size: {len(word_vectors.index_to_key)}")
# King - Man + Woman = ?
result = word_vectors.most_similar(positive=['king', 'woman'], negative=['man'], topn=5)
for word, similarity in result:
    print(f"{word}: {similarity:.4f}")
# Similar words to "computer"
print(word_vectors.most_similar('computer', topn=5))
# Odd one out
print(word_vectors.doesnt_match("breakfast lunch dinner banana".split()))
# Word similarity
print(word_vectors.similarity('king', 'queen'))
```

Output:

```
[==================================================] 100.0%
128.1/128.1MB downloaded
Vocabulary size: 400000
queen: 0.7699
monarch: 0.6843
throne: 0.6756
daughter: 0.6595
princess: 0.6521
[('computers', 0.8751984238624573), ('software', 0.8373122215270996), ('technology',
0.7642159461975098), ('pc', 0.7366448640823364), ('hardware', 0.7290390729904175)]
banana
0.7507691
```

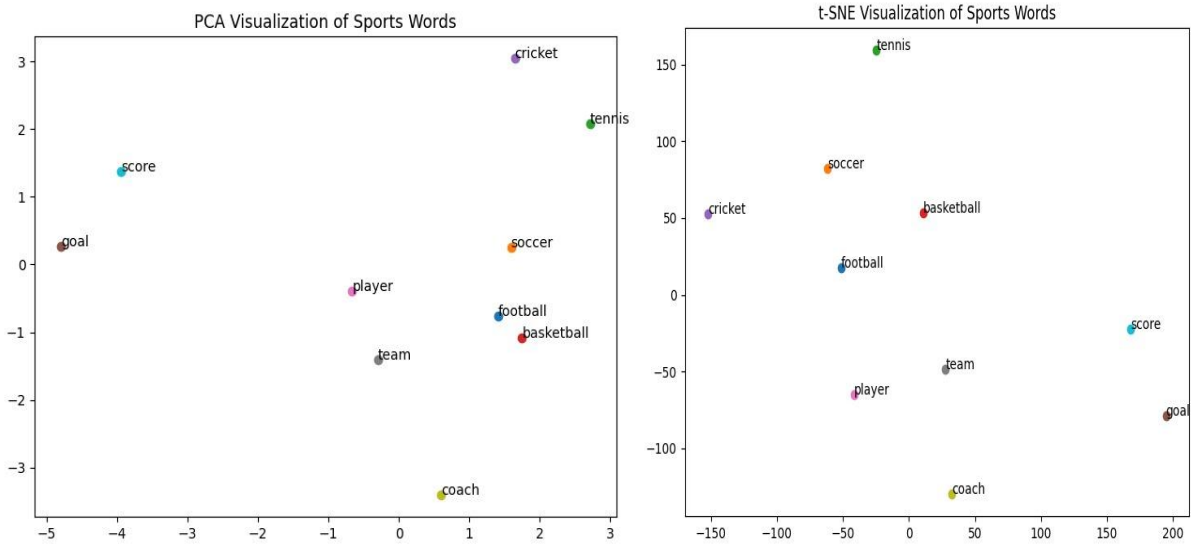
2.Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1. Select 10 words from aspecific domain (e.g., sports, technology) and visualize their embeddings. Analyze clusters and relationships.Generate contextually rich outputs using embeddings. Write a program to generate 5 semantically similar wordsfor a given input.

Source Code:

```
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE

# Sports words

sports_words = ['football', 'soccer', 'tennis', 'basketball', 'cricket', 'goal', 'player', 'team', 'coach', 'score']

sports_vectors = np.array([word_vectors[word] for word in sports_words])

# PCA Visualization

pca = PCA(n_components=2)

sports_2d = pca.fit_transform(sports_vectors)

plt.figure(figsize=(8,6))

for i, word in enumerate(sports_words):

plt.scatter(sports_2d[i,0], sports_2d[i,1])

plt.annotate(word, (sports_2d[i,0], sports_2d[i,1]))

plt.title("PCA Visualization of Sports Words")

plt.show()

tsne = TSNE(n_components=2, random_state=42, perplexity=5)

sports_tsne = tsne.fit_transform(sports_vectors)

plt.figure(figsize=(8,6))

for i, word in enumerate(sports_words):

plt.scatter(sports_tsne[i,0], sports_tsne[i,1])

plt.annotate(word, (sports_tsne[i,0], sports_tsne[i,1]))

plt.title("t-SNE Visualization of Sports Words")

plt.show()
```

```
# Function to get 5 semantically similar words
def get_similar_words(word):
    try:
        result = word_vectors.most_similar(word, topn=5)
        for w, sim in result:
            print(f"{w}: {sim:.4f}")
    except KeyError:
        print(f"'{word}' not in vocabulary!")
# Example
get_similar_words('football')
```

Output:



soccer: 0.8732

basketball: 0.8556

league: 0.8153

rugby: 0.8008

hockey: 0.7834

3.Train a custom Word2Vec model on a small dataset. Train embeddings on a domain-specific corpus (e.g., legal,medical) and analyze how embeddings capture domain-specific semantics.

Source Code:

```
medical_corpus = [
    "The patient was prescribed antibiotics to treat the bacterial infection.",
    "Diabetes mellitus is characterized by high blood sugar levels.",
    "MRI and CT scans are important tools for diagnosing brain and spinal cord injuries.",
    "Cardiovascular diseases include conditions like heart attack, stroke, and hypertension.",
    "Physical therapy helps patients recover mobility after surgery or injury.",
    "Vaccinations are critical in preventing infectious diseases such as measles and influenza.",
    "Common symptoms of flu include fever, cough, sore throat, and body aches.",
    "Blood pressure monitoring is vital for patients with hypertension or cardiovascular risks.",
    "Surgical removal of tumors requires precise planning and expert care.",
    "Medication dosages must be strictly followed to avoid adverse effects.",
    "Chronic kidney disease often requires dialysis or transplantation.",
    "Asthma is a respiratory condition causing difficulty in breathing due to airway
inflammation.",
    "The immune system protects the body against pathogens and foreign substances.",
    "Radiology departments use X-rays, ultrasounds, and MRIs for diagnostic imaging.",
    "Neurological disorders affect the brain, spinal cord, and nerves.",
    "Antiviral drugs are used to treat infections caused by viruses such as HIV and hepatitis.",
    "The doctor ordered blood tests to check for anemia and infection markers.",
    "Diuretics help reduce fluid buildup in patients with heart failure or kidney disease.",
    "Cholesterol levels impact the risk of developing atherosclerosis and heart disease.",
    "Emergency medical services provide urgent care for trauma and critical conditions.",]


from gensim.utils import simple_preprocess
from nltk.corpus import stopwords
import nltk
```

```
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

def preprocess(sentence):

    return [word for word in simple_preprocess(sentence) if word not in stop_words]

tokenized_corpus = [preprocess(sentence) for sentence in medical_corpus]

from gensim.models import Word2Vec

# Train Word2Vec with small parameters for demo

model = Word2Vec(sentences=tokenized_corpus, vector_size=50, window=3, min_count=1,
workers=2, epochs=100)

print("Most similar to 'disease':")

print(model.wv.most_similar('diseases', topn=5))

print("\nMost similar to 'blood':")

print(model.wv.most_similar('blood', topn=5))
```

Output:

```
Most similar to 'disease':

[('flu', 0.424752414226532), ('brain', 0.42404380440711975), ('departments',
0.42341890931129456), ('often', 0.4218001365661621), ('foreign', 0.42150747776031494)]

Most similar to 'blood':

[('helps', 0.5787495374679565), ('caused', 0.5051810145378113), ('patients',
0.49471670389175415), ('difficulty', 0.4769250154495239), ('removal',
0.4767632782459259)]
```

4.Use word embeddings to improve prompts for Generative AI model. Retrieve similar words using wordembeddings. Use the similar words to enrich a GenAI prompt. Use the AI model to generate responses for theoriginal and enriched prompts. Compare the outputs in terms of detail and relevance.

Source Code:

```
# Step 1: Pre-defined dictionary of words and their similar terms (static word
word_embeddings = {
    "ai": ["machine learning", "deep learning", "data science"],
    "data": ["information", "dataset", "analytics"],
    "science": ["research", "experiment", "technology"],
    "learning": ["education", "training", "knowledge"],
    "robot": ["automation", "machine", "mechanism"]
}
# Step 2: Function to find similar words using the static dictionary
def find_similar_words(word):
    if word in word_embeddings:
        return word_embeddings[word]
    else:
        return []
# Step 3: Function to enrich a prompt with similar words
def enrich_prompt(prompt):
    words = prompt.lower().split()
enriched_words = []
    for word in words:
similar_words = find_similar_words(word)
        if similar_words:
enriched_words.append(f"{word} ({', '.join(similar_words)})")
        else:
enriched_words.append(word)
    return " ".join(enriched_words)
```

```
# Step 4: Original prompt
original_prompt = "Explain AI and its applications in science."
# Step 5: Enrich the prompt using similar words
enriched_prompt = enrich_prompt(original_prompt)
# Step 2: Function to find similar words using the static dictionary
def find_similar_words(word):
    if word in word_embeddings:
        return word_embeddings[word]
    else:
        return []
# Step 3: Function to enrich a prompt with similar words
def enrich_prompt(prompt):
    words = prompt.lower().split()
enriched_words = []
    for word in words:
similar_words = find_similar_words(word)
        if similar_words:
enriched_words.append(f"{word} ({', '.join(similar_words)})")
        else:
enriched_words.append(word)
    return " ".join(enriched_words)
# Step 4: Original prompt
original_prompt = "Explain AI and its applications in science."
# Step 5: Enrich the prompt using similar words
enriched_prompt = enrich_prompt(original_prompt)
# Step 6: Print the original and enriched prompts
print("Original Prompt:")
print(original_prompt)
print("\nEnriched Prompt:")
print(enriched_prompt)
```

Output:

Original Prompt:

Explain AI and its applications in science.

Enriched Prompt:

explain ai (machine learning, deep learning, data science) and its applications in science.

5.Use word embeddings to create meaningful sentences for creative tasks. Retrieve similar words for a seed word.Create a sentence or story using these words as a starting point. Write a program that: Takes a seed word. Generatessimilar words. Constructs a short paragraph using these words.

Source Code:

```
# Step 1: Pre-defined dictionary of words and their similar terms
word_embeddings = {
    "adventure": ["journey", "exploration", "quest"],
    "robot": ["machine", "automation", "mechanism"],
    "forest": ["woods", "jungle", "wilderness"],
    "ocean": ["sea", "waves", "depths"],
    "magic": ["spell", "wizardry", "enchantment"]
}
# Step 2: Function to get similar words for a seed word
def get_similar_words(seed_word):
    if seed_word in word_embeddings:
        return word_embeddings[seed_word]
    else:
        return ["No similar words found"]
# Step 3: Function to create a short paragraph using the seed word and similar words
def create_paragraph(seed_word):
similar_words = get_similar_words(seed_word)
    if "No similar words found" in similar_words:
        return f"Sorry, I couldn't find similar words for '{seed_word}'."
 # Construct a short story using the seed word and similar words
    paragraph = (
f"Once upon a time, there was a great {seed_word}. "
f"It was full of {', '.join(similar_words[:-1])}, and {similar_words[-1]}. "
f"Everyone who experienced this {seed_word} always remembered it as a remarkable tale."
    )
```

```
    return paragraph
# Step 4: Input a seed word
seed_word = "adventure"  # You can change this to "robot", "forest", "ocean", "magic", etc.
# Step 5: Generate and print the paragraph
story = create_paragraph(seed_word)
print("Generated Paragraph:")
print(story)
```

Output:

Generated Paragraph:

Once upon a time, there was a great adventure. It was full of journey, exploration, and quest. Everyone who experienced this adventure always remembered it as a remarkable tale.

6.Use a pre-trained Hugging Face model to analyze sentiment in text. Assume a real-world application, Load thesentiment analysis pipeline. Analyze the sentiment by giving sentences to input.

Source Code:

```
from transformers import pipeline
# Load the sentiment-analysis pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
# Example feedback sentences (real-world inputs)
reviews = [
    "The product quality is amazing!",
    "I received the item.",
    "Customer service was terrible.",
    "I'm extremely satisfied with the delivery speed.",
    "The item broke after two days, very disappointed.",
    "Great value for money. Will buy again!"
]
# Analyze sentiment for each review
for review in reviews:
    result = sentiment_pipeline(review)[0] # returns a list with one dict
print(f"Review: \"{review}\"\n → Sentiment: {result['label']}, Score: {result['score']:.4f}\n")
```

Output:

```
Using a pipeline without specifying a model name and revision in production is not recommended.
Device set to use cpu
Review: "The product quality is amazing!"
 → Sentiment: POSITIVE, Score: 0.9999
Review: "I received the item."
```

→ Sentiment: POSITIVE, Score: 0.9996

Review: "Customer service was terrible."

→ Sentiment: NEGATIVE, Score: 0.9997

Review: "I'm extremely satisfied with the delivery speed."

→ Sentiment: POSITIVE, Score: 0.9994

Review: "The item broke after two days, very disappointed."

→ Sentiment: NEGATIVE, Score: 0.9996

Review: "Great value for money. Will buy again!"

→ Sentiment: POSITIVE, Score: 0.9997

7.Summarize long texts using a pre-trained summarization model using Hugging face model. Load thesummarization pipeline. Take a passage as input and obtain the summarized text.

Source Code:

```
# Step 1: Import the Hugging Face pipeline
from transformers import pipeline
# Step 2: Load the summarization pipeline
summarizer = pipeline("summarization")
# Step 3: Input a long passage for summarization
long_text = """
Artificial Intelligence (AI) is transforming various industries by automating tasks, improving efficiency,
and enabling new capabilities. In the healthcare sector, AI is used for disease diagnosis, personalized medicine,
and drug discovery. In the business world, AI-powered systems are optimizing customer service, fraud detection,
and supply chain management. AI's impact on everyday life is significant, from smart assistants to recommendation
systems in streaming platforms. As AI continues to evolve, it promises even greater advancements in fields like
education, transportation, and environmental sustainability.
"""
# Step 4: Summarize the input passage
summary = summarizer(long_text, max_length=50, min_length=20,
do_sample=False)[0]["summary_text"]
# Step 5: Print the summarized text
print("Summarized Text:")
print(summary)
```

Output:

Summarized Text:

Artificial Intelligence (AI) is transforming various industries by automating tasks, improving efficiency and enabling new capabilities . In healthcare sector, AI is used for disease diagnosis, personalized medicine, and drug discovery . In business world, AI-powered

8.Install langchain, cohere (for key), langchain-community. Get the apikey(By logging into Cohere and obtainingthe cohere key). Load a text document from your google drive . Create a prompt template to display the output ina particular manner.

Source Code:

```
# Step 1: Install necessary libraries
!pip install langchain cohere langchain-community# Step 2: Import the required modules
from langchain.llms import Cohere
from langchain.prompts import PromptTemplate
from langchain import LLMChain
from google.colab import drive
# Step 3: Mount Google Drive to access the document
drive.mount('/content/drive')
# Step 4: Load the text document from Google Drive
file_path = "/content/drive/MyDrive/Text/crow.txt"  # Change this path to your file location
with open(file_path, "r") as file:
    text = file.read()
# Step 5: Set up Cohere API key
cohere_api_key = "xNBCmAElkHo9M2tjr6d3SLP80NbMt8MofoUMsREE"
# Step 6: Create a prompt template
prompt_template = """
Summarize the following text in two bullet points:
{text}
"""
# Step 7: Configure the Cohere model with Langchain
llm = Cohere(cohere_api_key=cohere_api_key)
prompt = PromptTemplate(input_variables=["text"], template=prompt_template)
# Step 8: Create an LLMChain with the Cohere model and prompt template
chain = LLMChain(llm=llm, prompt=prompt)
# Step 9: Run the chain on the loaded text
```

```
result = chain.run(text)

print(text)

# Step 10: Display the formatted output

print("Summarized Output in Bullet Points:")

print(result)
```

Output:

In a spell of dry weather, when the Birds could find very little to drink, a thirsty Crow found a pitcher with a little water in it. But the pitcher was high and had a narrow neck, and no matter how he tried, the Crow could not reach the water. The poor thing felt as if he must die of thirst.

Then an idea came to him. Picking up some small pebbles, he dropped them into the pitcher one by one. With each pebble the water rose a little higher until at last it was near enough so he could drink.

Summarized Output in Bullet Points:

- During a dry spell, a crow found a pitcher with water inside, but the pitcher was too tall and narrow for the crow to reach the water.

- The crow dropped pebbles into the pitcher, raising the water level until it was reachable, saving the crow from thirst.

Would you like me to summarize any other texts?

9.Take the Institution name as input. Use Pydantic to define the schema for the desired output and create a customoutput parser. Invoke the Chain and Fetch Results. Extract the below Institution related details from Wikipedia:The founder of the Institution. When it was founded. The current branches in the institution . How manyemployees are working in it. A brief 4-line summary of the institution.

Source Code:

```
!pip install --quiet langchainpydanticwikipedia-apilangchain-core cohere langchain-community

from langchain_community.llms import Cohere

from langchain.prompts import PromptTemplate

from langchain_core.runnables import RunnableLambda

from pydantic import BaseModel

import wikipediaapi

# Schema for structured output

class InstitutionDetails(BaseModel):

    founder: str

    founded: str

    branches: str

    employees: str

    summary: str

# Limit wiki text to ~3000 characters (safe under token limit)

def fetch_wikipedia_summary(institution_name, max_chars=3000):

    wiki = wikipediaapi.Wikipedia(language='en', user_agent='InstitutionInfoBot/1.0 (https://www.wikipedia.org/)')

  page = wiki.page(institution_name)

    if not page.exists():

        return "No information available."

    return page.text[:max_chars]  # truncate long text

# Prompt template
```

```
prompt_template = """
Extract the following information from the given text:
- Founder
- Founded (year)
- Current branches
- Number of employees
- 4-line brief summary
Text: {text}
Format:
Founder: <founder>
Founded: <founded>
Branches: <branches>
Employees: <employees>
Summary: <summary>
"""
# Main logic
if _name_ == "_main_":
institution_name = input("Enter the name of the institution: ")
wiki_text = fetch_wikipedia_summary(institution_name)
 # Replace with your real key, and keep it secret in production
llm = Cohere(cohere_api_key="cHb51fJ9urTonXJaiG6l7sOGUoMseDK6tMGz8mRN")
prompt = PromptTemplate.from_template(prompt_template)
    chain = prompt | llm  # ✅ new way: pipe prompt into llm


    # Get output
    response = chain.invoke({"text": wiki_text})
# print("\nRaw LLM Response:\n", response)
 # Try to parse into structured form
    try:
        lines = response.strip().split('\n')
```

```
        info = {line.split(':')[0].lower(): ':'.join(line.split(':')[1:]).strip() for line in lines if ':' in
line}
        details = InstitutionDetails(
            founder=info.get("founder", "N/A"),
            founded=info.get("founded", "N/A"),
            branches=info.get("branches", "N/A"),
            employees=info.get("employees", "N/A"),
            summary=info.get("summary", "N/A")
        )
print("\nInstitution Details:")
print(f"Founder: {details.founder}")
print(f"Founded: {details.founded}")
print(f"Branches: {details.branches}")
print(f"Employees: {details.employees}")
print(f"Summary: {details.summary}")
    except Exception as e:
print("Error parsing response:",e)
```

Output:

Enter the name of the institution: google

Institution Details:

Founder: Larry Page and Sergey Brin

Founded: September 4, 1998

Branches: American multinational corporation

Employees: Unknown

Summary: Google LLC is an American multinational corporation and technology company focusing on online advertising, search engine technology, cloud computing, computer software, quantum computing, e-commerce, consumer electronics, and artificial intelligence. Google was founded on September 4, 1998, by American computer scientists Larry Page and Sergey Brin while they were PhD students at Stanford University in California. Together, they own about 14% of its publicly listed shares and control 56% of its stockholder voting powerthrough super-voting stock

10.Build a chatbot for the Indian Penal Code. We'll start by downloading the official Indian Penal Code document,and then we'll create a chatbot that can interact with it. Users will be able to ask questions about the Indian PenalCode and have a conversation with it.

Source Code:

```
# Step 1: Install necessary packages

!pip install langchain pydantic wikipedia-api openai

# Step 2: Import required modules

from langchain.chains import load_qa_chain

from langchain.docstore.document import Document

from langchain.llms import OpenAI

# Step 3: Load the Indian Penal Code text from a file

ipc_file_path = "path_to_your_ipc_file.txt" # Replace with the actual path to your IPC text
file

# Read the IPC document

with open(ipc_file_path, "r", encoding="utf-8") as file:

 ipc_text = file.read()

# Step 4: Create a Langchain Document object

ipc_document = Document(page_content=ipc_text)

# Step 5: Set up OpenAI (or any other LLM of your choice)

llm = OpenAI(openai_api_key="YOUR_OPENAI_API_KEY", temperature=0.3) # Use

temperature=0.3 for more factual responses

# Step 6: Create a simple question-answering chain

qa_chain = load_qa_chain(llm, chain_type="stuff")

# Step 7: Chat with the chatbot

print("Chatbot for the Indian Penal Code (IPC)")

print("Ask a question about the Indian Penal Code (type 'exit' to stop):")

while True:

 user_question = input("\nYour question: ")

 if user_question.lower() == "exit":
```

```
print("Goodbye!")
break
# Use the QA chain to answer the question
response = qa_chain.run(input_documents=[ipc_document], question=user_question)
print(f"Answer: {response}"
```

Output:

Chatbot for the Indian Penal Code (IPC) Ask a question about the Indian Penal Code (type 'exit' to stop):

Your question: What is Section 302 of the IPC?

Answer: Section 302 of the Indian Penal Code refers to punishment for murder,

which is punishable with death or life imprisonment and a fine.

Your question: exit

Goodbye!