HKBK College of Engineering

(Affiliated to Visvesvaraya Technological University Belgaum and approved by AICTE, New Delhi and Govt.of Karnataka)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELGAVI-590018, KARNATAKA

LABORATORY MANUAL

Generative AI

BAI657C

(Effective from the academic year 2024 -2025)

Prepared By:

Prof. SHRUTHI V KULKARNI

Verified By

DQAC	HOD

HKBK COLLEGE OF ENGINEERING

Opp to Manyata Techpark,
Nagwara, Bangalore – 560 045

HKBK COLLEGE OF ENGINEERING
Established in 1997

Department of Artificial Intelligence and Machine Learning

Vision and Mission of the Institution

Vision

To empower students through wholesome education and enable the students to develop into highly qualified and trained professionals with ethics and emerge as responsible citizens with a broad outlook to build a vibrant nation.

Mission

M1. To achieve academic excellence in science, engineering, and technology through dedication to duty, innovation in teaching, and faith in human values.

M2. To enable our students to develop into outstanding professionals with high ethical standards to face the challenges of the 21st century.

M3. To provide educational opportunities to the deprived and weaker section of society to uplift their socioeconomic status.

Vision and Mission of the AIML Department

Vision

To advance the intellectual capacity of the nation and the international community by imparting knowledge to graduates who are globally recognized as innovators, entrepreneurs and competent professionals.

Mission

M1. To provide excellent technical knowledge and computing skills to make the graduates globally competitive with professional ethics.

M2. To be involved in research activities and be committed to lifelong learning to positive contributions to society.

Programme Educational Objectives

PEO-1	To provide students with a strong foundation in engineering fundamentals and in the computer science and engineering to work in the global scenario.
PEO-2	To provide sound knowledge of programming and computing techniques and good communication and interpersonal skills so that they will be capable of analyzing, designing and building innovative software systems.
PEO-3	To equip students in the chosen field of engineering and related fields to enable him to work in multidisciplinary teams.
PEO-4	To inculcate in students professional, personal and ethical attitude to relate engineering issues to broader social context and become responsible citizen.
PEO-5	To provide students with an environment for life-long learning which allow them to successfully adapt to the evolving technologies throughout their professional carrier and face the global challenges.

Programme Outcomes	
a	**Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
b	**Problem Analysis: Identify,** formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
c	**Design/ Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
d	**Conduct investigations of complex problems** using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.
e	**Modern Tool Usage**: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an under- standing of the limitations.
f	**The Engineer and Society**: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.
g	**Environment and Sustainability**: Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.
h	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.

i.	**Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams and in multidisciplinary settings.
j.	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
k	**Life-long Learning**: Recognize the need for and have the preparation and ability to engage in independent and life- long learning in the broadest context of technological change.
l.	**Project Management and Finance**: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
Programme Specific Outcomes	
m	**Problem-Solving Skills:** An ability to investigate and solve a problem by analysis, interpretation of data, design and implementation through appropriate techniques, tools and skills.
n	**Professional Skills**: An ability to apply algorithmic principles, computing skills and computer science theory in the modelling and design of computer-based systems.
o	**Entrepreneurial Ability:** An ability to apply design, development principles and management skills in the construction of software product of varying complexity to become an entrepreneur

HKBK COLLEGE OF ENGINEERING
Established in 1997

Department of Artificial Intelligence and Machine Learning

Academic Year: 2024-2025 Semester: 6TH EVEN

LAB MANUAL

Subject: **Generative AI** Subject Code: BAI657C

For the Period From: Feb 2025 To: May 2025

Course Learning Objectives:

- Understand the principles and concepts behind generative AI models
- Explain the knowledge gained to implement generative models using Prompt design frameworks.
- Apply various Generative AI applications for increasing productivity.
- Develop Large Language Model-based Apps.

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Develop the ability to explore and analyze word embeddings, perform vector arithmetic to investigate word relationships, visualize embeddings using dimensionality reduction techniques
- Apply prompt engineering skills to real-world scenarios, such as information retrieval, text generation.
- Utilize pre-trained Hugging Face models for real-world applications, including sentiment analysis and text summarization.
- Apply different architectures used in large language models, such as transformers, and understand their advantages and limitations.

List of Experiments

Sl No.	Experiments
1.	Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyze results.
2.	Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1. Select 10 words from a specific 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 words for 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 word embeddings. Use the similar words to enrich a GenAI prompt. Use the AI model to generate responses for the original 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. Generates similar words. 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 the sentiment 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 the summarization pipeline. Take a passage as input and obtain the summarized text.
8.	Install langchain, cohere (for key), langchain-community. Get the api key(By logging into Cohere and obtaining the cohere key). Load a text document from your google drive . Create a prompt template to display the output in a 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 many employees 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 Penal Code and have a conversation with it.

1. Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyse results.

Program:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk import FreqDist
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')
corpus = ['king is a strong man','queen is a wise woman','boy is a young man',
        'girl is a young woman','prince is a young','prince will be strong',
        'princess is young','man is strong','woman is pretty','prince is a boy',
        'prince will be king','princess is a girl','princess will be queen']
statements_list = [word_tokenize(cor) for cor in corpus]
print(f"\n\nstatements_list = {statements_list}")


stop_words = set(stopwords.words('english'))
documents = [[word for word in document if word.lower() not in stop_words] for document
in statements_list]
print(f'\n\ndocuments = {documents}')


vocabulary = FreqDist([word for document in documents for word in document])
print(f'\n\nvocabulary = {vocabulary}')


vector1 = vocabulary['king']
print(vector1)
vector2 = vocabulary['man']
print(vector2)
sum_vector = vector1 + vector2
diff_vector = vector1 - vector2
print(f"\nsum vector = {sum_vector}")
print(f"\ndifference vector = {diff_vector}")
```

```
from nltk import Text

similarity = (vocabulary['king'] * vocabulary['queen']) / (vocabulary['king'] ** 0.5 *
vocabulary['queen'] ** 0.5)

print(f"\nCosine Similarity between 'king' and 'queen': {similarity}")


most_similar = sorted(vocabulary.items(), key=lambda item: item[1], reverse=True)

print(f"\nMost Similar words to 'king': {most_similar}")


analogy_vector = vocabulary['king'] - vocabulary['man'] + vocabulary['woman']

def distance(item):

    return abs(item[1] - analogy_vector)

most_similar_analogy = sorted(vocabulary.items(), key=distance)[1]

print(f"\nAnalogy Result (king - man + woman): {most_similar_analogy}")
```

Output:

True

statements_list = [['king', 'is', 'a', 'strong', 'man'], ['queen', 'is', 'a', 'wise', 'woman'], ['boy', 'is', 'a', 'young', 'man'], ['girl', 'is', 'a', 'young', 'woman'], ['prince', 'is', 'a', 'young'], ['prince', 'will', 'be', 'strong'], ['princess', 'is', 'young'], ['man', 'is', 'strong'], ['woman', 'is', 'pretty'], ['prince', 'is', 'a', 'boy'], ['prince', 'will', 'be', 'king'], ['princess', 'is', 'a', 'girl'], ['princess', 'will', 'be', 'queen']]

documents = [['king', 'strong', 'man'], ['queen', 'wise', 'woman'], ['boy', 'young', 'man'], ['girl', 'young', 'woman'], ['prince', 'young'], ['prince', 'strong'], ['princess', 'young'], ['man', 'strong'], ['woman', 'pretty'], ['prince', 'boy'], ['prince', 'king'], ['princess', 'girl'], ['princess', 'queen']]

vocabulary = <FreqDist with 12 samples and 30 outcomes>

2
3

sum vector = 5

difference vector = -1

Cosine Similarity between 'king' and 'queen': 1.9999999999999996

Most Similar words to 'king': [('young', 4), ('prince', 4), ('strong', 3), ('man', 3), ('woman', 3), ('princess', 3), ('king', 2), ('queen', 2), ('boy', 2), ('girl', 2), ('wise', 1), ('pretty', 1)]

Analogy Result (king - man + woman): ('queen', 2)

2. Use dimensionality reduction (e.g., PCA or t-SNE) to visualize word embeddings for Q 1. Select 10 words from a specific 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 words for a given input.

Program:

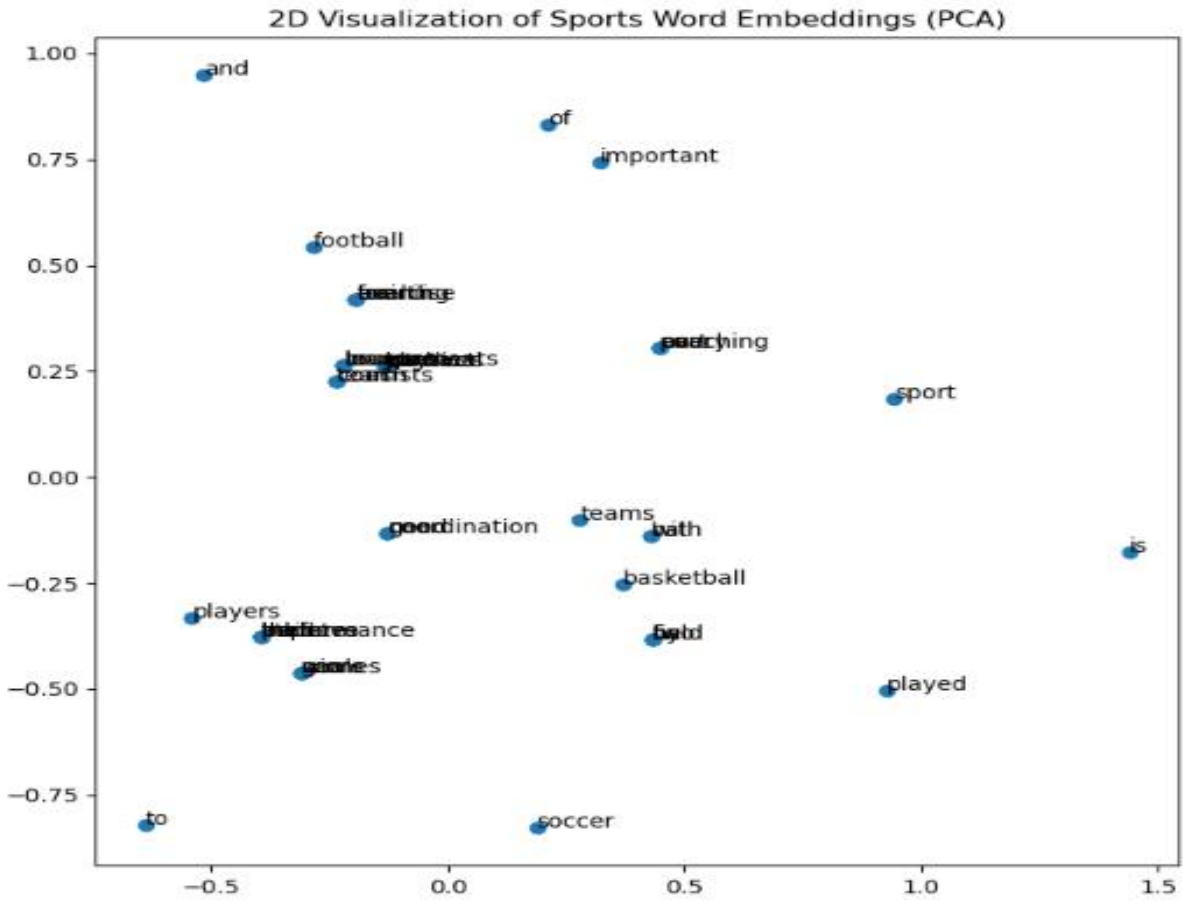
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import PCA
from sklearn.metrics.pairwise import cosine_similarity

# Step 1: Create a custom sports-related corpus
corpus = [
    'basketball is a sport played with a ball',
    'soccer is played by two teams on a field',
    'football involves a lot of physical contact',
    'athletes train hard to improve their performance',
    'coaching is an important part of every sport',
    'basketball players need good coordination',
    'a team consists of players and a coach',
    'training and exercise are important for health',
    'soccer players score goals to win games',
    'football teams compete in leagues and tournaments'
]

# Step 2: Convert text into word embeddings using CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus).toarray()
words = vectorizer.get_feature_names_out()
print(words)

# Step 3: Reduce dimensionality to 2D using PCA
pca = PCA(n_components=2)
word_vectors_2d = pca.fit_transform(X.T)  # Transpose to get words instead of sentences


# Step 4: Find similar words using cosine similarity
def find_similar_words(target_word, word_vectors, words, top_n=5):
    if target_word not in words:
        return f"'{target_word}' not found in vocabulary."

    target_idx = np.where(words == target_word)[0][0]
    target_vector = word_vectors[target_idx].reshape(1, -1)

    similarities = cosine_similarity(target_vector, word_vectors)[0]
    similar_word_indices = similarities.argsort()[::-1][1:top_n+1]  # Exclude the word itself

    similar_words = [words[i] for i in similar_word_indices]
    return similar_words
input_word = "basketball"
similar_words = find_similar_words(input_word, word_vectors_2d, words, top_n=5)
print(f"Words similar to '{input_word}': {similar_words}")
```

```
# Step 5: Visualize word embeddings
plt.figure(figsize=(8, 8))
plt.scatter(word_vectors_2d[:, 0], word_vectors_2d[:, 1])

for i, word in enumerate(words):
    plt.text(word_vectors_2d[i, 0], word_vectors_2d[i, 1], word, fontsize=10)

plt.title("2D Visualization of Sports Word Embeddings (PCA)")
plt.show()
```

Output:

Tokenized corpus: [['basketball', 'is', 'a', 'sport', 'played', 'with', 'a', 'ball'], ['soccer', 'is', 'played', 'by', 'two', 'teams', 'on', 'a', 'field'], ['football', 'involves', 'a', 'lot', 'of', 'physical', 'contact'], ['athletes', 'train', 'hard', 'to', 'improve', 'their', 'performance'], ['coaching', 'is', 'an', 'important', 'part', 'of', 'every', 'sport'], ['basketball', 'players', 'need', 'good', 'coordination'], ['a', 'team', 'consists', 'of', 'players', 'and', 'a', 'coach'], ['training', 'and', 'exercise', 'are', 'important', 'for', 'health'], ['soccer', 'players', 'score', 'goals', 'to', 'win', 'games'], ['football', 'teams', 'compete', 'in', 'leagues', 'and', 'tournaments']]

Words similar to 'basketball': ['played', 'field', 'on', 'two', 'by']



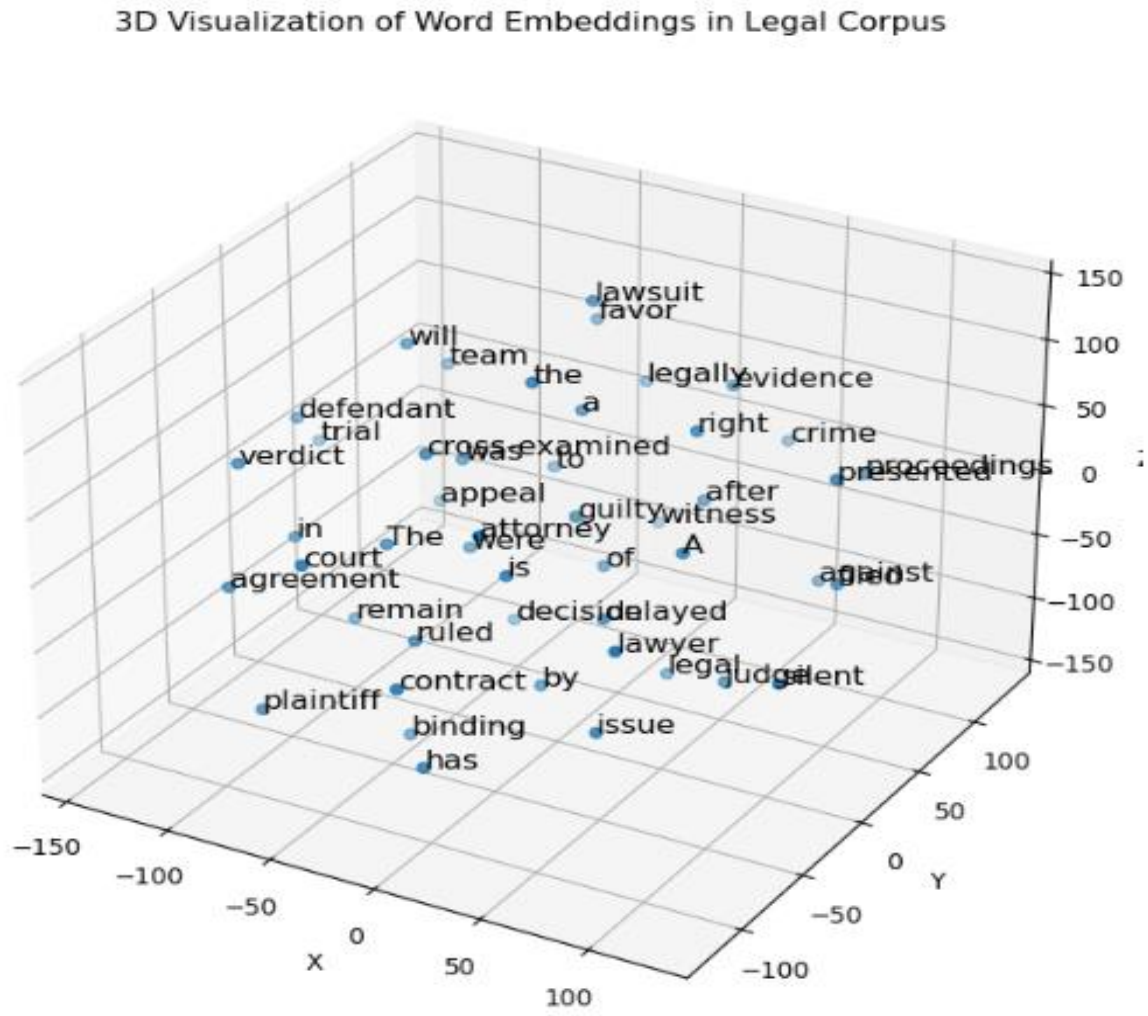
3. Train a custom Word2Vec model on a small dataset. Train embeddings on a domain-specific corpus (e.g., legal, medical) and analyze how embeddings capture domain-specific semantics.

Program:

```
import gensim
from gensim.models import Word2Vec
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Step 1: Create a small legal-related corpus
legal_corpus = [
    "The defendant is guilty of the crime",
    "The plaintiff filed a lawsuit against the defendant",
    "The court ruled in favor of the defendant",
    "The legal proceedings were delayed by the judge",
    "The lawyer presented evidence in court",
    "A judge will issue a verdict after the trial",
    "The defendant has the right to remain silent",
    "A contract is a legally binding agreement",
    "The witness was cross-examined by the attorney",
    "The legal team will appeal the decision"
]

# Step 2: Tokenize the corpus (splitting into words)
corpus_tokenized = [sentence.split() for sentence in legal_corpus]

# Step 3: Train the Word2Vec model on the legal corpus
model = Word2Vec(corpus_tokenized, min_count=1, vector_size=100, window=5)

# Step 4: Extract word embeddings
words = list(model.wv.index_to_key)  # List of words in the vocabulary
word_embeddings = np.array([model.wv[word] for word in words])
```

```
# Step 5: Reduce dimensions to 3 for visualization (using PCA or t-SNE)
# Use PCA to reduce from 100 dimensions to 47 (the number of words in the corpus)
pca = PCA(n_components=47)
word_embeddings_pca = pca.fit_transform(word_embeddings)

# Then, reduce from 47 dimensions to 3 using t-SNE for better visualization
tsne = TSNE(n_components=3, random_state=42)
word_embeddings_tsne = tsne.fit_transform(word_embeddings_pca)

# Step 6: Plot the 3D visualization
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot each word in 3D space
ax.scatter(word_embeddings_tsne[:, 0], word_embeddings_tsne[:, 1], word_embeddings_tsne[:,
2])

# Annotate the points with the corresponding words
for i, word in enumerate(words):
    ax.text(word_embeddings_tsne[i, 0], word_embeddings_tsne[i, 1], word_embeddings_tsne[i,
2], word, fontsize=12)

# Set labels for the axes
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Title of the plot
ax.set_title('3D Visualization of Word Embeddings in Legal Corpus')

# Show the plot
plt.show()
```

Output:


3D Visualization of Word Embeddings in Legal Corpus
lawsuit
favor
will
team
the
legally
evidence
a
defendant
trial
right
crime
verdict
cross-examined
was
to
proceedings
presented
appeal
guilty
after
witness
in
The
were
attorney
court
A
agreement
is
of
remain
decision
delayed
ruled
lawyer
legal
judge
silent
contract
by
plaintiff
binding
issue
has
X
Y
150
100
50
0
−50
−100
−150

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

Program:

```
from gensim.models import Word2Vec
import random

# Simulate word embeddings using a predefined dictionary
word_embeddings = {
    'defendant': ['argued', 'verdict', 'suing', 'judge', 'case'],
    'court': ['judge', 'lawyer', 'trial', 'evidence', 'jury'],
    'justice': ['fairness', 'law', 'rights', 'punishment', 'order']
}

# Function to retrieve similar words
def get_similar_words(word, topn=5):
    return word_embeddings.get(word, [])[:topn]

# Function to enrich the prompt
def enrich_prompt(original_prompt, similar_words):
    return f"{original_prompt} Include terms like {', '.join(similar_words)} to provide more
detail."

# Simulate AI response generation
def generate_response(prompt):
    return f" AI response for prompt: '{prompt}' "

# Define original prompt
original_prompt = "Explain the role of a defendant in a court case."

# Retrieve similar words and enrich the prompt
similar_words = get_similar_words('defendant')
enriched_prompt = enrich_prompt(original_prompt, similar_words)
```

```
# Generate responses
original_response = generate_response(original_prompt)
enriched_response = generate_response(enriched_prompt)


# Compare responses
print("Most similar words to 'defendant':", similar_words)
print("\nResponse for \033[1m Original Prompt:\033[0m\n", original_response)
print("\nResponse for \033[1m Enriched Prompt:\033[0m\n", enriched_response)
```

Output:

```
Most similar words to 'defendant': ['argued', 'verdict', 'suing', 'judge', 'case']

Response for Original Prompt:
 AI response for prompt: 'Explain the role of a defendant in a court case.'

Response for Enriched Prompt:
 AI response for prompt: 'Explain the role of a defendant in a court case. Include terms like argue
d, verdict, suing, judge, case to provide more detail.'
```

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. Generates similar words. Constructs a short paragraph using these words.

Program:

```
import gensim.downloader as api
import random
import nltk
from nltk.tokenize import sent_tokenize
# Ensure required resources are downloaded
nltk.download('punkt')


# Load pre-trained word vectors
print("Loading pre-trained word vectors...")
word_vectors = api.load("glove-wiki-gigaword-100")  # 100D GloVe word embeddings
print("Word vectors loaded successfully!")
def get_similar_words(seed_word, top_n=5):
    """Retrieve top-N similar words for a given seed word."""
    try:
        similar_words = word_vectors.most_similar(seed_word, topn=top_n)
        return [word[0] for word in similar_words]
    except KeyError:
        print(f"'{seed_word}' not found in vocabulary. Try another word.")
        return []
def generate_sentence(seed_word, similar_words):
    """Create a meaningful sentence using the seed word and its similar words."""
    sentence_templates = [
        f"The {seed_word} was surrounded by {similar_words[0]} and {similar_words[1]}.",
        f"People often associate {seed_word} with {similar_words[2]} and {similar_words[3]}.",
        f"In the land of {seed_word}, {similar_words[4]} was a common sight.",
        f"A story about {seed_word} would be incomplete without {similar_words[1]} and
{similar_words[3]}.",
        ]
    return random.choice(sentence_templates)
```

```
def generate_paragraph(seed_word):
    """Construct a creative paragraph using the seed word and similar words."""
    similar_words = get_similar_words(seed_word, top_n=5)
    if not similar_words:
        return "Could not generate a paragraph. Try another seed word."
    paragraph = [generate_sentence(seed_word, similar_words) for _ in range(4)]
    return " ".join(paragraph)


# Example usage
seed_word = input("Enter a seed word: ")
model = api.load("glove-wiki-gigaword-100")
paragraph = generate_paragraph(seed_word)
print("\nGenerated Paragraph:\n")
print(paragraph)
```

Output:

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Admin\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Loading pre-trained word vectors...
Word vectors loaded successfully!
Enter a seed word:  ocean

Generated Paragraph:

A story about ocean would be incomplete without waters and coast. The ocean was surrounded b
y sea and waters. People often associate ocean with seas and coast. A story about ocean would be
incomplete without waters and coast.
```

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

Program:

```
!pip install –q transformers
!pip3 install emoji==0.6.0

# Using pipeline class to make predictions from models available in the Hugging Face
#transformers library


from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
data = ["Traffic was horrible today, & ruined my entire mood.", "The movie was fantastic! I
would highly recommend it."]
sentiment_pipeline(data)

# Using a specific model for sentiment analysis

specific_model = pipeline(model="finiteautomata/bertweet-base-sentiment-analysis")
specific_model(data)
```

Output:

```
Device set to use cpu
[{'label': 'NEGATIVE', 'score': 0.9998144507408142},
{'label': 'POSITIVE', 'score': 0.999872088432312}]

Device set to use cpu
[{'label': 'NEG', 'score': 0.9833412766456604},
{'label': 'POS', 'score': 0.9922917485237122}]
```

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

Program:

```
# Install transformers if not already installed
!pip install -q transformers

# Import the required library
from transformers import pipeline

# Load the summarization pipeline (uses a pretrained model like BART or T5)
summarizer = pipeline("summarization")

# Define a long passage
OriginalText = """
Generative AI continues to evolve rapidly, influencing various sectors and prompting discussions on its applications and implications. The development of multimodal AI models capable of processing and generating multiple forms of data, such as text, images, and audio, is accelerating. These models enhance the versatility and applicability of AI systems across various domains, from interactive storytelling to comprehensive data analysis. Experts predict that by 2027, a significant portion of generative AI solutions will be multimodal, reflecting a shift towards more integrated and sophisticated AI capabilities. Generative AI is enhancing AR and VR experiences by creating more realistic and interactive environments. This includes procedural content generation for virtual worlds and the development of interactive non-player characters (NPCs) that can engage users with unscripted dialogues and actions, thereby elevating the level of immersion in virtual experiences. These trends underscore the dynamic nature of generative AI and its expanding influence across various sectors. As the technology progresses, it remains crucial to address the associated ethical, legal, and infrastructural challenges to fully harness its potential.
"""
print("")
print("Original Text:")
print(OriginalText)

# Summarize the text
summary = summarizer(OriginalText, max_length=100, min_length=30, do_sample=False)

# Print the summarized text
print("Summarized Text:")
print(summary[0]['summary_text'])
```

Output:

No model was supplied, defaulted to sshleifer/distilbart-cnn-12-6 and revision a4f8f3e (https://huggingface.co/sshleifer/distilbart-cnn-12-6).
Using a pipeline without specifying a model name and revision in production is not recommended.
Device set to use cpu

Original Text:

Generative AI continues to evolve rapidly, influencing various sectors and prompting discussions on its applications and implications. The development of multimodal AI models capable of processing and generating multiple forms of data, such as text, images, and audio, is accelerating. These models enhance the versatility and applicability of AI systems across various domains, from interactive storytelling to comprehensive data analysis. Experts predict that by 2027, a significant portion of generative AI solutions will be multimodal, reflecting a shift towards more integrated and sophisticated AI capabilities. Generative AI is enhancing AR and VR experiences by creating more realistic and interactive environments. This includes procedural content generation for virtual worlds and the development of interactive non-player characters (NPCs) that can engage users with unscripted dialogues and actions, thereby elevating the level of immersion in virtual experiences. These trends underscore the dynamic nature of generative AI and its expanding influence across various sectors. As the technology progresses, it remains crucial to address the associated ethical, legal, and infrastructural challenges to fully harness its potential.

Summarized Text:

The development of multimodal AI models capable of processing and generating multiple forms of data, such as text, images, and audio, is accelerating. Generative AI is enhancing AR and VR experiences by creating more realistic and interactive environments.

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

Program:

```
#  Step 1: Install required libraries (Run this only once)
!pip install langchain cohere langchain-community google-colab

#  Step 2: Import necessary libraries
import cohere
import getpass
from langchain import PromptTemplate
from langchain.llms import Cohere
from google.colab import auth
from google.colab import drive

# Import necessary libraries for Google Drive API
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseDownload
import io

#  Step 3: Authenticate Google Drive
auth.authenticate_user()
drive.mount('/content/gdrive')

#  Step 4: Load the Text File from Google Drive
file_path = '/content/drive/MyDrive/textfileForpgm8.txt'

# Initialize text_content to an empty string
text_content = ""

try:
  # Create Google Drive API client
    drive_service = build('drive', 'v3')
    request = drive_service.files().get_media(fileId=file_id)
    fh = io.BytesIO()
    downloader = MediaIoBaseDownload(fh, request)
    done = False
    while done is False:
        status, done = downloader.next_chunk()

    # Decode the downloaded content and store it
    text_content = fh.getvalue().decode('utf-8')
    print("✅ File loaded successfully!")

except Exception as e:
  print("❌ Error loading file:", str(e))


#  Step 5: Set Up Cohere API Key
COHERE_API_KEY = getpass.getpass("🔑 Enter your Cohere API Key: ")
```

```
#  Step 6: Initialize Cohere Model with LangChain
cohere_llm = Cohere(cohere_api_key=COHERE_API_KEY, model="command")


# Step 7: Create a Prompt Template
template = """
You are an AI assistant helping to summarize and analyze a text document.
Here is the document content:
{text}
🔹 Summary:
- Provide a concise summary of the document.
🔹 Key Takeaways:
- List 3 important points from the text.
🔹 Sentiment Analysis:
- Determine if the sentiment of the document is Positive, Negative, or
Neutral.
"""
prompt_template = PromptTemplate(input_variables=["text"],
template=template)

#  Step 8: Format the Prompt and Generate Output
formatted_prompt = prompt_template.format(text=text_content)
response = cohere_llm.predict(formatted_prompt)
#  Step 9: Display the Generated Output
print("\n📌 **Formatted Output** 📌")
print(response)
```

Steps to get Cohere API Key:

Step 1: Go to Cohere website
Step 2: Sign in with your Gmail or Github
Step 3: On left side panel of the Cohere Dashboard – Click API Keys
Step 4: Navigate to Trial Keys (Free) – Click Create Trial Key
Step 5: Give the name for the Key – Click Generate trail key
Step 6: Copy the Cohere API Key & use it while execution

Output:

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).

❌ Error loading file: <HttpError 404 when requesting https://www.googleapis.com/drive/v3/files/textfileForpgm8.txt?alt=media returned "File not found: textfileForpgm8.txt.". Details: "[{'message': 'File not found: textfileForpgm8.txt.', 'domain': 'global', 'reason': 'notFound', 'location': 'fileId', 'locationType': 'parameter'}]">

🔑 Enter your Cohere API Key: · · · · · · · · · ·

📌 **Formatted Output** 📌

Here is a summary of the text you provided:

The text is a set of instructions for an AI assistant to summarize and analyze a text document. Based on the instructions provided, it can be inferred that the text document will be related to this specific set of instructions.

The key takeaways from these instructions are:

1. An AI assistant is asked to summarize and analyse a text document efficiently

2. The summary should be concise

3. The sentiment of the document needs to be determined, and it should be one of the three following options: positive, neutral, or negative.

The sentiment of the text document is Neutral because it is a concise set of instructions and does not convey an emotional tone or opinion toward a particular subject.

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 many employees are working in it. A brief 4-line summary of the institution.

Program:

```
# Install required libraries
!pip install wikipedia-api pydantic
from pydantic import BaseModel
from typing import List, Optional
import wikipediaapi
class InstitutionDetails(BaseModel):
  founder: Optional[str]
  Type: Optional[str]
  Established: Optional[str]
  branches: Optional[List[str]]
  number_of_employees: Optional[int]
  summary: Optional[str]
def fetch_institution_details(institution_name: str) ->
InstitutionDetails:
# Define a user-agent as per Wikipedia's policy
  user_agent = "Colab Notebooks/ (contact: myemail@hkbk.edu.in)"
  wiki_wiki = wikipediaapi.Wikipedia(user_agent=user_agent, language='en')
  page = wiki_wiki.page(institution_name)
  if not page.exists():
      raise ValueError(f"The page for '{institution_name}' does not exist
on Wikipedia.")
# Initialize variables
  founder = None
  Type = None
  Established = None
  branches = []
  number_of_employees = None
# Extract summary
  summary = page.summary[:1000] # Limiting summary to 1000 characters
# Extract information from the infobox
  infobox = page.text.split('\n')
  for line in infobox:
      if 'Founder' in line:
          founder = line.split(':')[-1].strip()
      elif 'Type' in line:
          Type = line.split(':')[-1].strip()
      elif 'Established' in line:
          Established = line.split(':')[-1].strip()
      elif 'Branches' in line:
          branches = [branch.strip() for branch in line.split(':')[-
1].split(',')]
      elif 'Number of employees' in line:
          try:
              number_of_employees = int(line.split(':')[-
1].strip().replace(',',''))
```

```
            except ValueError:
                number_of_employees = None
    return InstitutionDetails(
        founder=founder,
        Type=Type,
        Established=Established,
        branches=branches if branches else None,
        number_of_employees=number_of_employees,
        summary=summary
    )
# Import necessary libraries
from IPython.display import display
import ipywidgets as widgets
# Function to display institution details
def display_institution_details(details: InstitutionDetails):
    print(f"Founder: {details.founder or 'N/A'}")
    print(f"Type: {details.Type or 'N/A'}")
    print(f"Established: {details.Established or 'N/A'}")
    print(f"Branches: {', '.join(details.branches) if details.branches
else 'N/A'}")
    print(f"Number of Employees: {details.number_of_employees or 'N/A'}")
    print(f"Summary: {details.summary or 'N/A'}")
# Function to handle button click
def on_button_click(b):
    institution_name = text_box.value
    try:
        details = fetch_institution_details(institution_name)
        display_institution_details(details)
    except ValueError as e:
        print(e)
# Create input box and button
text_box = widgets.Text(
    value='',
    placeholder='Enter the institution name',
    description='Institution:',
    disabled=False
)
button = widgets.Button(
    description='Fetch Details',
    disabled=False,
    button_style='',
    tooltip='Click to fetch institution details',
    icon='search'
)
# Set up button click event
button.on_click(on_button_click)
# Display input box and button
display(text_box, button)
```

Output:

Institution: Indian Institute of
Fetch Details

Founder: N/A
Type: N/A
Established: N/A
Branches: N/A
Number of Employees: N/A
Summary: The Indian Institute of Science (IISc) is a public, deemed, research university for higher education and research in science, engineering, design, and management. It is located in Bengaluru, Karnataka. The institute was established in 1909 with active support from Jamsetji Tata and thus is also locally known as the Tata Institute. It was granted a deemed university status in 1958 and recognized as an Institute of Eminence in 2018.

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 Penal Code and have a conversation with it.

Program:

```
#this library is designed for efficient extraction of text, images, and
metadata from PDFs
!pip install pymupdf

 # This will open a file selector
from google.colab import files
uploaded = files.upload()

#lists all files in the current working directory (this will list the file
uploaded.)
!ls

# Import necessary libraries
import fitz  # PyMuPDF
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

# Make sure the file is correctly uploaded or the path is valid
pdf_path = 'IPC.pdf'

# Open the PDF and extract text
doc = fitz.open(pdf_path)
ipc_text = ""
for page_num in range(doc.page_count):
    page = doc.load_page(page_num)
    ipc_text += page.get_text()
doc.close()

# Preprocess the text: split into sentences
sentences = ipc_text.split(".")
sentences = [s.strip() for s in sentences if s.strip()]

# Load the Sentence-BERT model for sentence embeddings
model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

# Generate embeddings for all sentences in the IPC document
ipc_embeddings = model.encode(sentences)

# Function to query the IPC document
def query_ipc_document(query, top_k=3):
    # Embed the user's query
    query_embedding = model.encode([query])

    # Calculate cosine similarity between the query and IPC sentences
    similarities = cosine_similarity(query_embedding, ipc_embeddings)
```

```
    # Get the top k most similar sentences
    top_k_indices = similarities[0].argsort()[-top_k:][::-1]
    top_k_sentences = [sentences[i] for i in top_k_indices]

    return top_k_sentences

# Test the query function
query = input("Ask a question about the Indian Penal Code: ")
answer = query_ipc_document(query)

print("\nHere are the top responses from the IPC document:\n")
for idx, sentence in enumerate(answer):
    print(f"{idx+1}. {sentence}")
```

Output:

Choose Files IPC.pdf

- **IPC.pdf**(application/pdf) - 1104850 bytes, last modified: 4/3/2025 - 100% done

Saving IPC.pdf to IPC.pdf

IPC.pdf sample_data

Ask a question about the Indian Penal Code: What is the punishment for sales of adulterated drugs?

Here are the top responses from the IPC document:

1. Sale of adulterated drugs
2. Sale of adulterated drugs
3. —Whoever, knowing any drug or medical preparation to have been adulterated in such a manner as to lessen its efficacy, to change its operation, or to render it noxious, sells the same, or offers or exposes it for sale, or issues it from any dispensary for medicinal purposes as unadulterated, or causes it to be used for medicinal purposes by any person not knowing of the adulteration, shall be punished with imprisonment of either description for a term which may extend to six months, or with fine which may extend to one thousand rupees, or with both