# BANGALORE INSTITUE OF TECHNOLOGY

Krishna Rajendra Rd, Parvathipuram, Vishweshwarapura,
Basavanagudi, Bengaluru, Karnataka 560004

Affiliated to

**Visveswaraya Technological University**

**Department of CSE-(IOT,& Cyber Security including Block Chain Technology)**

**LAB MANUAL**

**Course Name: Generative AI**

**Course Code: BAIL657C**

**VI Semester**

**2022 Scheme**

## PROGRAM 1

**Explore pre-trained word vectors. Explore word relationships using vector arithmetic. Perform arithmetic operations and analyze results.**

**Word vectors :** A word vector is a list of numbers that represents a word's meaning, based on how that word is used in different contexts.Computers don't understand words like "king" or "apple" as we do. So we turn each word into a **vector**—a fixed-size list of numbers—based on its meaning and usage in sentences.

Here are the some of the Examples :
king → [0.8, 0.3, 0.6]
queen → [0.8, 0.4, 0.6]
man → [0.7, 0.1, 0.5]
woman → [0.7, 0.2, 0.5]

**Vector Arithmetic** : Vector arithmetic in word vectors is using operations like addition and subtraction to explore semantic relationships between words.

Example : "Paris" - "France" + "Italy" ≈ "Rome" .
. "King" - "Man" ≈ "Royalty" (or the essence of kingship)

**Requirement**: Install gensim model using : **" pip install gensim "**

**GENSIM:** Gensim is an open-source Python library that uses topic modelling and document similarity modelling to manage and analyse massive amounts of unstructured text data. It is especially well-known for applying topic and vector space modelling algorithms, such as Word2Vec and Latent Dirichlet Allocation (LDA), which are widely used.

**CODE:**

```python
import gensim.downloader as api

# Load pre-trained word vectors (GloVe 50D)
model = api.load("glove-wiki-gigaword-50")

# Lowercase words (GloVe model expects lowercase)
word1 = "paris"
word2 = "france"
word3 = "india"
word4 = "capital"

# Vector math
vector_add = model[word1] + model[word4]                # Paris + capital
vector_mix = model[word1] - model[word2] + model[word3]    # paris - france + india

# Get most similar words
result_add = model.most_similar([vector_add], topn=1)
result_mix = model.most_similar([vector_mix], topn=1)

# Display the results
print(f"The result of '{word2} + {word4}' is: {result_add[0][0]}")
print(f"The result of '{word1} - {word2} + {word3}' is: {result_mix[0][0]}")
```

**OUTPUT :**
```
The result of 'Paris + capital' is: france
 The result of 'paris - france + india' is: delhi
```

## PROGRAM 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.**

**Word embeddings :** They are numerical representations of words in the form of vectors, where words with similar meanings have similar values. They help computers understand the relationships between words based on how they appear together in large amounts of text.

**t-SNE** (pronounced "tee-snee")-(***t-distributed Stochastic Neighbor Embedding***) is a tool used to visualize high-dimensional data (like word embeddings or image features) in 2D or 3D. In Generative AI, data like word vectors or image features often exist in 100+ dimensions, which we can't see. t-SNE helps **turn that complex data into a visual map**, where similar things are placed close together, and different things are far apart.

For example: Words like *cat*, *dog*, and *tiger* will appear near each other.
. Words like *apple* or *car* will appear in different areas.

**t-SNE** might give better clusters than **PCA** because it captures non-linear structures.

**`glove-wiki-gigaword-50`:** It is a pre-trained word embedding model with **50-dimensional vectors** trained on a large corpus combining Wikipedia and Gigaword 5. It represents words as numerical vectors capturing basic semantic relationships, and is ideal for lightweight NLP tasks due to its smaller size and fast loading .

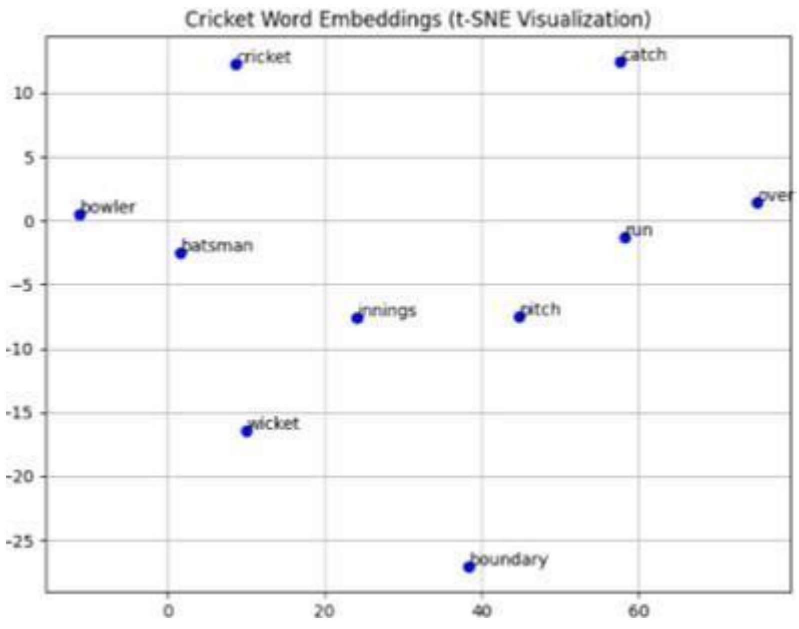
Requirement : pip install scikit-learn

**CODE :**

```
import gensim.downloader as api
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np

# Load small pre-trained GloVe model
print("Loading word vectors...")
model = api.load("glove-twitter-50")
print("Model loaded.")

# Cricket-related words to visualize
words = ['wicket', 'innings', 'bowler', 'batsman', 'over', 'run', 'pitch',
'boundary', 'catch', 'cricket']

# Keep only words that exist in the model
valid_words = [word for word in words if word in model]
word_vectors = np.array([model[word] for word in valid_words])

# Use t-SNE to reduce dimensions for plotting
tsne = TSNE(n_components=2, perplexity=5, random_state=0)
word_vectors_2d = tsne.fit_transform(word_vectors)

# Create a blank plot of size 8x6 inches
plt.figure(figsize=(8, 6))
```

```
# Loop through each word and its 2D position
for i in range(len(valid_words)):
    x = word_vectors_2d[i][0]
    y = word_vectors_2d[i][1]

    plt.scatter(x, y, color='blue')                                        #
Plot each word as a blue dot
    plt.text(x + 0.1, y + 0.1, valid_words[i])                   # Label the
dot with the word

plt.title("Cricket Word Embeddings (t-SNE Visualization)")
plt.grid(True)
plt.show()



# Find and print similar words to 'cricket'
if 'cricket' in model:
    print("\nWords most similar to 'cricket':")
    for similar_word, score in model.most_similar('cricket', topn=5):
        print(f"{similar_word:15} {score:.3f}")
else:
    print("'cricket' not found in the vocabulary.")
```

**OUTPUT:**



```
Words most similar to 'cricket':
ipl          0.796
lanka        0.777
dhoni        0.755
england      0.755
batsman      0.741
```

## PROGRAM 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**

**Word Embendings** : In NLP ,it is a term used for representation of words for text analysis, typically in form of a real vaalued vector that encodes the meaning of the word such that they are closer in the vector space are expanded to be in similar meaning.

**Need for Word Embedding?** To reduce dimensionality
. To use a word to predict the words around it.
. Inter-word semantics must be captured.

**Word2Vec** : Devolped by Google ,widely used method in natural language processing (NLP) that allows words to be represented as vectors in a continuous vector space.
Main Principle : Words with similar meanings should have similar vector representations.
Word2Vec utilizes two architectures: **1. CBOW (Continuous Bag of Words)** and **2. Skip Gram.**

**NLTK:** For handling human language data, NLTK, or Natural Language Toolkit, is a potent Python library. It offers user-friendly interfaces to more than 50 lexical resources and corpora, including WordNet. A collection of text processing libraries for tasks like categorization, tokenization, stemming, tagging, parsing, and semantic reasoning are also included with NLTK.

CODE:

```python
# Import necessary libraries
import gensim
from gensim.models import Word2Vec
import nltk
from nltk.tokenize import word_tokenize

# Download tokenizer data (only needs to be run once)-optional
nltk.download('punkt')

# Sample medical-related text data
corpus = [
    "A patient with diabetes requires regular insulin injections.",
    "Medical professionals recommend exercise for heart health.",
    "Doctors use MRI scans to diagnose brain disorders.",
    "Antibiotics help fight bacterial infections but not viral infections.",
    "The surgeon performed a complex cardiac surgery successfully.",
    "Doctors and nurses work together to treat patients.",
    "A doctor specializes in diagnosing and treating diseases."
]

# Step 1: Tokenize the text (convert sentences to lowercase word lists)
tokenized_corpus = [word_tokenize(sentence.lower()) for sentence in corpus]

# Step 2: Train the Word2Vec model
model = Word2Vec(
    sentences=tokenized_corpus,
vector_size=100,    # Size of word vectors
    window=3,           # Context window size
    min_count=1,        # Include all words
    workers=4,          # Use 4 threads
    sg=1                # Use Skip-gram (sg=0 for CBOW)
)

# Step 3: Save the model for later use
model.save("medical_word2vec.model")

# Step 4: Find top 5 words most similar to 'doctor'
similar_words = model.wv.most_similar("doctor", topn=5)

# Step 5: Display the results neatly
print("\nWords most similar to 'doctor':")
 for i, (word, similarity) in enumerate(similar_words, start=1):
     print(f"{i}. {word}  (Similarity Score: {similarity:.2f})")
```

**OUTPUT:**

```
Words most similar to 'doctor':
1. heart  (Similarity Score: 0.24)
2. surgeon  (Similarity Score: 0.20)
3. injections  (Similarity Score: 0.16)
4. health  (Similarity Score: 0.16)
5. patients  (Similarity Score: 0.15)
```

## PROGRAM 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.**

**Requirements :** pip install transformers gensim

**Transformers**: A library by Hugging Face that provides state-of-the-art pre-trained models (like BERT, GPT, etc.) for tasks like text generation, classification, translation, and more using simple APIs like pipeline().

**Gensim**: A powerful library for unsupervised topic modeling and natural language processing, known for working with **word embeddings** like Word2Vec, GloVe, and FastText; api is used to load pre-trained models easily.

**CODE :**

```python
from transformers import pipeline
import gensim.downloader as api

# Load pre-trained GloVe embeddings
glove_model = api.load("glove-wiki-gigaword-50")

# Retrieve similar words to a key term
word = "technology"
similar_words = glove_model.most_similar(word, topn=5)
print(f"Similar words to '{word}': {similar_words}")

# Load a text generation pipeline using GPT-2
generator = pipeline("text-generation", model="gpt2")

# Function to generate text from a prompt
def generate_response(prompt, max_length=100):
    response = generator(prompt, max_length=max_length, num_return_sequences=1)
    return response[0]['generated_text']

# Original prompt
original_prompt = "Explain the impact of technology on society."
original_response = generate_response(original_prompt)

# Enriched prompt using similar words from the embedding
enriched_prompt = (
    "Explain the impact of technology, innovation, science, "
    "engineering, and digital advancements on society."
)
enriched_response = generate_response(enriched_prompt)

# Print responses
print("Original Prompt Response:")
print(original_response)
print("Enriched Prompt Response:")
print(enriched_response)
```

**OUTPUT :**

```
Similar words to 'technology': [('technologies', 0.8928266167640686), ('computer', 0.8525559306144714), ('systems', 0.8289327025413513), ('software', 0.8089751601219177), ('computing', 0.7991364002227783)]
```

## PROGRAM 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.**

**CODE:**

```python
import gensim.downloader as gensim_api

# Load GloVe word vectors
glove_model = gensim_api.load("glove-wiki-gigaword-50")

# Function to build a tech-related narrative
def generate_paragraph(core_word, related_words):
    passage = (
        f"In a world shaped by {core_word}, innovation starts with a spark of
{related_words[0][0]}. "
        f"Each step in {related_words[1][0]} reveals breakthroughs that redefine
{related_words[2][0]}. "
        f"Engineers and thinkers collaborate to transform every {related_words[3][0]}
into a leap toward "
        f"{related_words[4][0]}."
    )
    return passage

# New seed word
main_word = "technology"
top_matches = glove_model.most_similar(main_word, topn=5)

# Create and display the paragraph
generated_output = generate_paragraph(main_word, top_matches)
print(generated_output)
```

**OUTPUT:**

```
In a world shaped by technology, innovation starts with a spark of technologies. Each step in computer reveals breakthroughs
that redefine systems. Engineers and thinkers collaborate to transform every software into a leap toward computing.
```

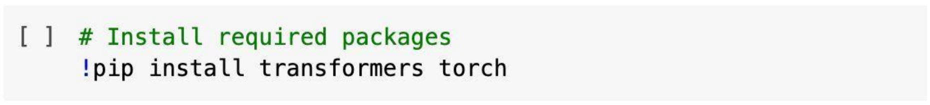
## PROGRAM 6

**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.**

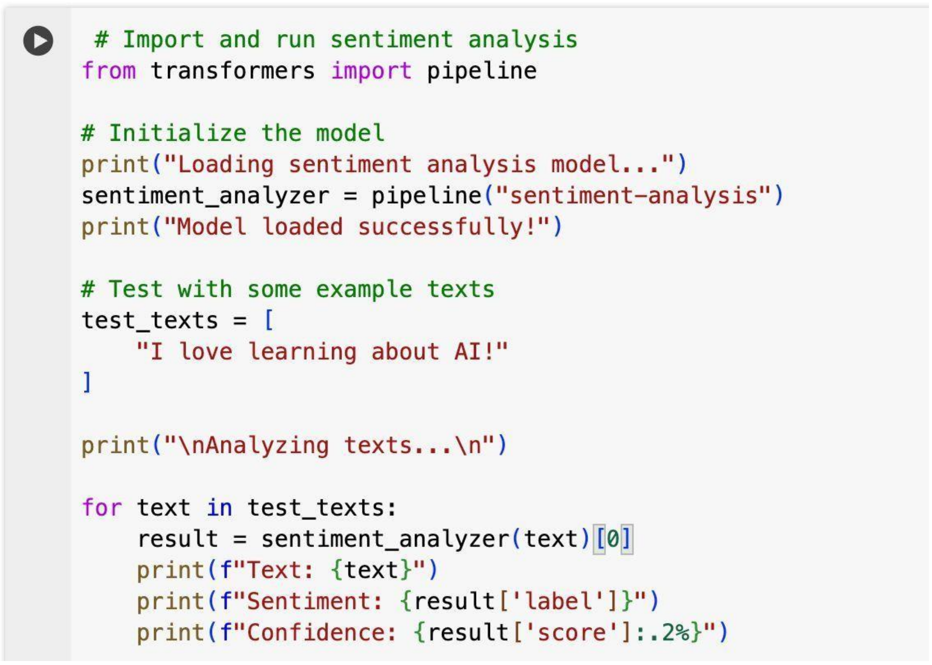
**Requirements:** !pip install transformers torch

**Hugging Face :**
Hugging Face is an open-source AI platform known for its Transformers library, which provides pre-trained models for various NLP tasks. Using its simple pipeline API, developers can perform sentiment analysis, text classification, and more with minimal code.

**Code:**

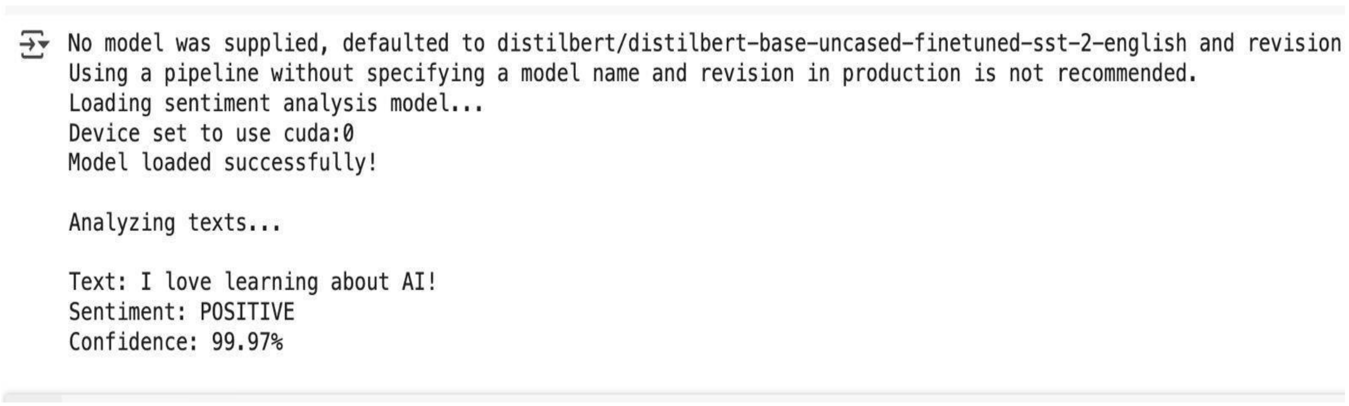


```
[ ] # Install required packages
    !pip install transformers torch
```



```python
 # Import and run sentiment analysis
from transformers import pipeline

# Initialize the model
print("Loading sentiment analysis model...")
sentiment_analyzer = pipeline("sentiment-analysis")
print("Model loaded successfully!")

# Test with some example texts
test_texts = [
    "I love learning about AI!"
]

print("\nAnalyzing texts...\n")

for text in test_texts:
    result = sentiment_analyzer(text)[0]
    print(f"Text: {text}")
    print(f"Sentiment: {result['label']}")
    print(f"Confidence: {result['score']:.2%}")
```

**Output:**



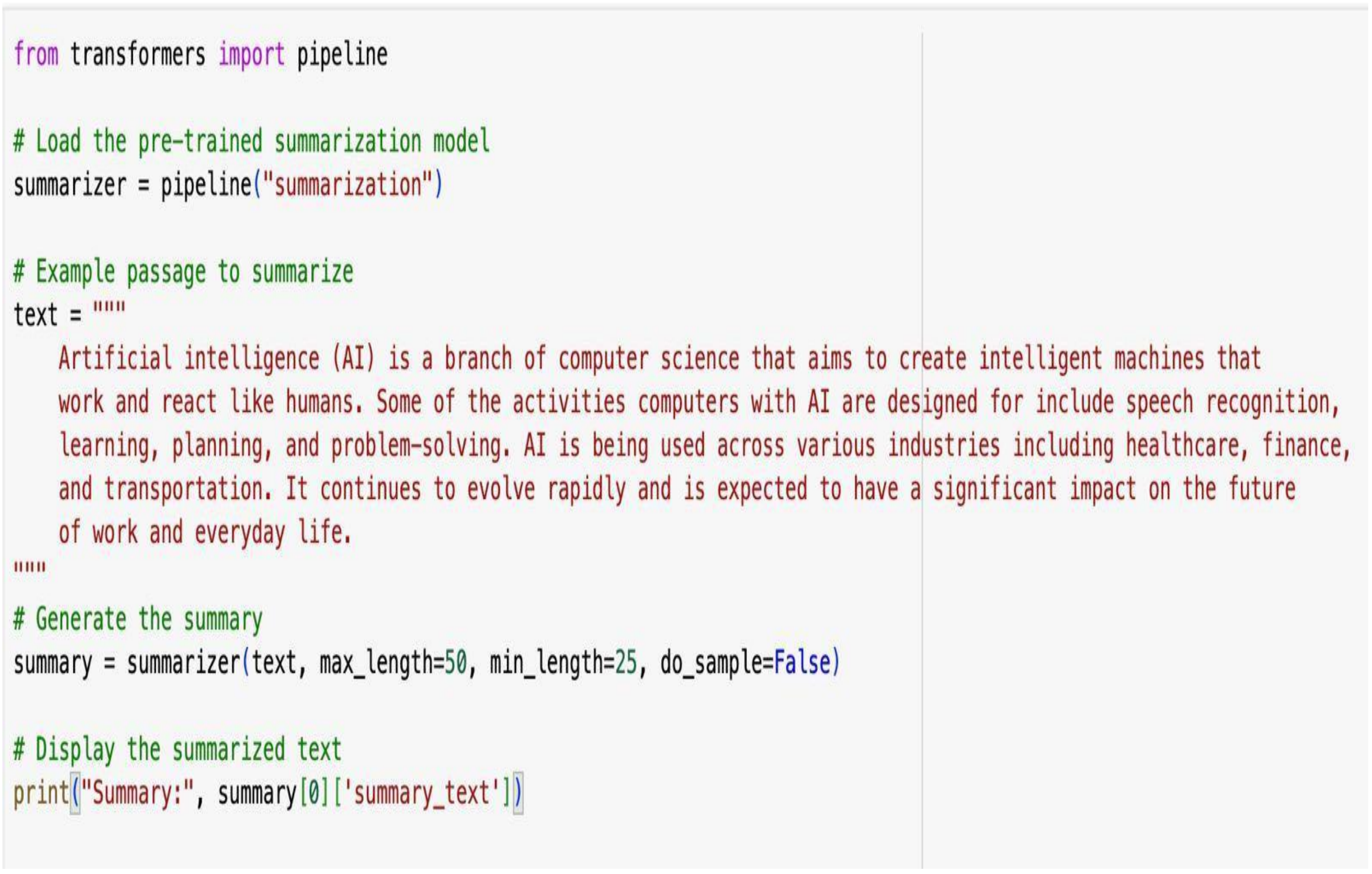
```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision
Using a pipeline without specifying a model name and revision in production is not recommended.
Loading sentiment analysis model...
Device set to use cuda:0
Model loaded successfully!

Analyzing texts...

Text: I love learning about AI!
Sentiment: POSITIVE
Confidence: 99.97%
```

## PROGRAM 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.**

**Requirements:** !pip install transformers

**Code:**



```
from transformers import pipeline

# Load the pre-trained summarization model
summarizer = pipeline("summarization")

# Example passage to summarize
text = """
    Artificial intelligence (AI) is a branch of computer science that aims to create intelligent machines that
    work and react like humans. Some of the activities computers with AI are designed for include speech recognition,
    learning, planning, and problem-solving. AI is being used across various industries including healthcare, finance,
    and transportation. It continues to evolve rapidly and is expected to have a significant impact on the future
    of work and everyday life.
"""
# Generate the summary
summary = summarizer(text, max_length=50, min_length=25, do_sample=False)

# Display the summarized text
print("Summary:", summary[0]['summary_text'])
```

**Output:**

Summary: Artificial intelligence (AI) is a branch of computer science that aims to create intelligent machines that work and react like humans. AI is being used across various industries, including healthcare, finance, and transportation.

## PROGRAM 8

**Install Langchain, Cohere (for API key), and Langchain-community. Get the API key (by logging into Cohere). Load a text document from Google Drive. Create a prompt template to display the output in a particular manner**

**Theory**

- LangChain: A framework for building applications powered by LLMs (Large Language Models).
- Cohere: Provides state-of-the-art NLP models via API for text generation, classification, and embeddings.
- Prompt Templates: Pre-defined formats that guide the model's responses in a structured way.

**Requirements**

1. Install required libraries:

    pip install langchain langchain-community cohere google-auth google-auth-oauthlib google-auth-httplib2 googleapiclient

2. Get a Cohere API Key:

    Sign up at https://cohere.com/ and generate an API key.

3. Enable Google Drive API:

    Obtain credentials for Google Drive API to access text files.

**Code:**

```python
import os
from cohere import Client
from langchain.prompts import PromptTemplate

# Step 1: Set Cohere API Key
os.environ["COHERE_API_KEY"] = "Generate your own API key"
co = Client(os.getenv("COHERE_API_KEY"))

# Step 2: Load Text Document (Option 1: Local File)
with open(r"C:\Users\Admin\Documents\AI.txt", "r", encoding="utf-8") as file:
    text_document = file.read()

# Step 3: Create a Prompt Template
template = """
You are an expert summarizer. Summarize the following text in a concise manner:
Text: {text}
Summary:
"""
prompt_template = PromptTemplate(input_variables=["text"], template=template)
formatted_prompt = prompt_template.format(text=text_document)

# Step 4: Send Prompt to Cohere API
response = co.generate(
    model="command",
    prompt=formatted_prompt,
    max_tokens=40
)
```

```python
# Step 5: Display Output
print("Summary:")
print(response.generations[0].text.strip())
```

## OUTPUT

AI is a diverse set of technologies that emulate human cognitive functions, resulting in various applications like machine learning, chatbots, and autonomous vehicles. Its rapid development

## PROGRAM 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 Institution-related details from Wikipedia (founder, founding year, branches, employees, summary).**

### Theory

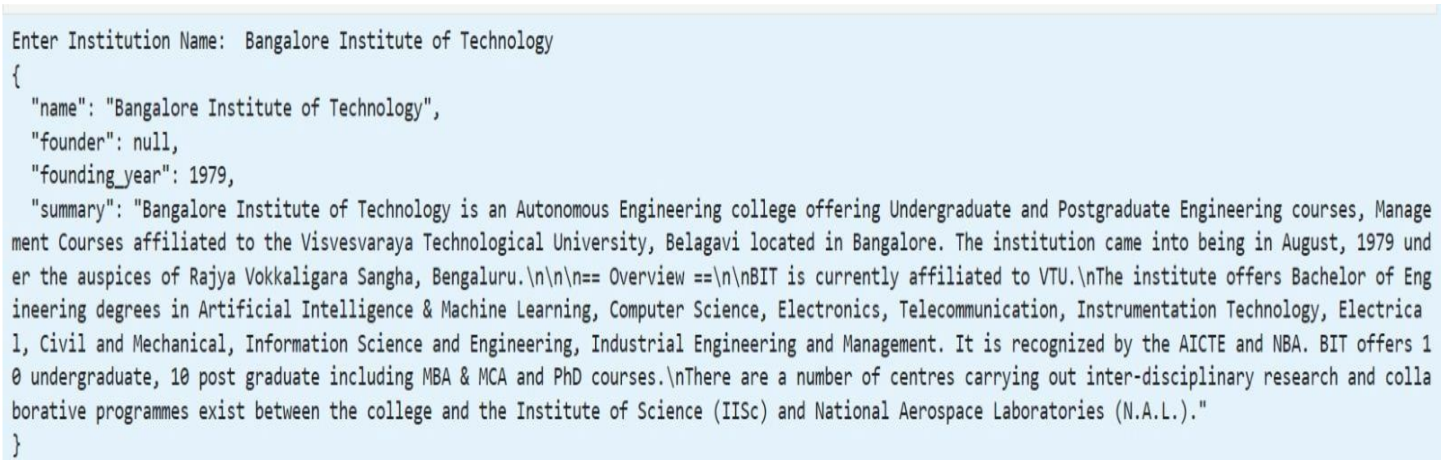
- Pydantic: A Python library for **data validation** and defining schemas.
- **LangChain**: Helps **process natural language queries** using LLMs.
- **Wikipedia API**: Fetches structured information from Wikipedia.

**REQUIREMENTS:** !pip install Wikipedia
!pip install pydantic

### CODE

```python
import wikipedia
from pydantic import BaseModel
from typing import Optional
import json

# Pydantic schema for the output
class InstitutionInfo(BaseModel):
    name: str
    founder: Optional[str]
    founding_year: Optional[int]
    summary: str

# Function to fetch Wikipedia details
def get_institution_info(name: str) -> Optional[InstitutionInfo]:
    try:
        page_title = wikipedia.search(name)[0]
        page = wikipedia.page(page_title)
        content = page.content.lower()
        summary = wikipedia.summary(page_title, sentences=5)

        # Extract details from content
        founder = next((line.split("founded by")[1].split('.')[0].strip().title() for 
line in content.split('\n') if "founded by" in line), None)
        year = next((int(word) for word in content.split() if word.isdigit() and 
len(word) == 4), None)

        return InstitutionInfo(name=page.title, founder=founder, founding_year=year, 
summary=summary)

    except Exception as e:
        print(f"Error: {e}")
        return None

# Main logic
if _name_ == "_main_":
    name = input("Enter Institution Name: ")
    info = get_institution_info(name)
    if info:
        print(json.dumps(info.dict(), indent=2))
```

**OUTPUT:**



```
Enter Institution Name:  Bangalore Institute of Technology
{
  "name": "Bangalore Institute of Technology",
  "founder": null,
  "founding_year": 1979,
  "summary": "Bangalore Institute of Technology is an Autonomous Engineering college offering Undergraduate and Postgraduate Engineering courses, Manage
ment Courses affiliated to the Visvesvaraya Technological University, Belagavi located in Bangalore. The institution came into being in August, 1979 und
er the auspices of Rajya Vokkaligara Sangha, Bengaluru.\n\n\n== Overview ==\n\nBIT is currently affiliated to VTU.\nThe institute offers Bachelor of Eng
ineering degrees in Artificial Intelligence & Machine Learning, Computer Science, Electronics, Telecommunication, Instrumentation Technology, Electrica
l, Civil and Mechanical, Information Science and Engineering, Industrial Engineering and Management. It is recognized by the AICTE and NBA. BIT offers 1
0 undergraduate, 10 post graduate including MBA & MCA and PhD courses.\nThere are a number of centres carrying out inter-disciplinary research and colla
borative programmes exist between the college and the Institute of Science (IISc) and National Aerospace Laboratories (N.A.L.)."
}
```

## PROGRAM 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.**

**Theory:**
A chatbot is an AI-powered application designed to simulate human conversation. For this task, we use Retrieval-Based NLP, where the chatbot fetches relevant sections from a document instead of generating responses from scratch.

**Key Concepts Involved:**
1. Natural Language Processing (NLP) – Used for text processing, query understanding, and response generation.
2. Document Retrieval – Searching and extracting relevant sections from the IPC text.
3. Embeddings & Similarity Search – Techniques like TF-IDF, Word2Vec, or LLMs (e.g., GPT) to find the most relevant section for a given query.
4. Chatbot Frameworks – Using LangChain, ChromaDB, or Hugging Face Transformers for better response handling.

**Requirements:** Install required libraries:
pip install streamlit langchain langchain-community langchain-core langchain-textsplitters langchain-ollama pdfplumber

DOWNLOAD THE PDF FROM THE GIVEN LINK :
https://www.indiacode.nic.in/bitstream/123456789/15289/1/ipc_act.pdf
Load the pdf into the directory to get the output

**CODE:**

```
 from PyPDF2 import PdfReader
 import re

# Extract text from IPC PDF (adjusted path)
text = "".join([page.extract_text() for page in PdfReader("ipc.pdf").pages[5:50]])

# Split into sections
sections = [s.replace('\n', ' ').strip() for s in
re.findall(r'(\d+\..*?)(?=\n\d+\.|\Z)', text, re.DOTALL)]

# IPC Bot
def ipc_bot(q):
    q = q.lower()
    for sec in sections:
        if (match := re.search(r'section\s*(\d+)', q)) and
sec.startswith(f"{match.group(1)}."):
            return sec.split('.')[1].strip().split('.')[0] + "."
        if any(word in sec.lower() for word in q.split()):
            return sec.split('.')[1].strip().split('.')[0] + "."
    return " No match."
```

```
# Chat loop
while True:
    q = input(" You: ")
    if q.lower() in ['exit', 'quit']: break
    print("IPC Bot:", ipc_bot(q))
```