Pgm1:

```
!pip install --upgrade gensim scipy
from gensim.models.fasttext import load_facebook_model
model_path="/kaggle/input/cc-en-300-bin/cc.en.300.bin"
fasttext_model=load_facebook_model(model_path)
print("Fasttext Loaded successful")

print(fasttext_model.wv['prince'])

from gensim.models import KeyedVectors
model_path="/kaggle/input/google-word2vec/GoogleNews-vectors-negative300.bin"
word2vec_model=KeyedVectors.load_word2vec_format(model_path, binary=True)
print("Google 2 vec loaded")

print(word2vec_model['queen'])

king=fasttext_model.wv["king"]
queen=fasttext_model.wv["queen"]
man=fasttext_model.wv["man"]
woman=fasttext_model.wv["woman"]

print("King vector:",king[:5])
print("Queen vector:",queen[:5])
new_vector=king+woman
similar_words=fasttext_model.wv.similar_by_vector(new_vector,topn=5)
print("Closet words to (king+woman):",similar_words)
```

Pgm2:

```
!pip install --upgrade gensim scipy
from gensim.models.fasttext import load_facebook_model
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
model_path="/kaggle/input/cc-en-300-bin/cc.en.300.bin"
fasttext_model=load_facebook_model(model_path)
print("fasttext model loaded successfully!")
king=fasttext_model.wv["king"]
queen=fasttext_model.wv["queen"]
print("king vector: ",king[:5])
```

```
print("queen vector: ",queen[:5])
tech_words=['Lawyer','Judge','Court','Fruit','Quantum','Encryption','da
tabase','computernetworks','Cybersecurity','Artificialintelligence']
for i in tech_words:
    print(i)
word_vectors=np.array([fasttext_model.wv[word] for word in tech_words])
word_vectors[0],word_vectors[1]
pca=PCA(n_components=2)
embeddings_2d=pca.fit_transform(word_vectors)
plt.figure(figsize=(10,8))
plt.scatter(embeddings_2d[:,0],embeddings_2d[:,1],marker='o',color='blu
e')
for i,word in enumerate(tech_words):

plt.annotate(word,(embeddings_2d[i,0],embeddings_2d[i,1]),fontsize=12)
plt.title("2D PCA projection of technology word embeddings")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.grid()
plt.show
import gensim
word_vectors =
gensim.models.KeyedVectors.load_word2vec_format("/kaggle/input/google-w
ord2vec/GoogleNews-vectors-negative300.bin",binary=True)
def find_similar_words(word, top_n=5):
    try:
        similar_words = word_vectors.most_similar(word, topn=top_n)
        return similar_words
    except KeyError:
        return f"The word '{word}' is not in the vocabulary."
word = "king"
similar_words = find_similar_words(word)
print(f"Top 5 similar words to '{word}':")
for sim_word, similarity in similar_words:
    print(f"{sim_word}: {similarity}")
```

Pgm3:

```
!pip install gensim nltk
import nltk
from nltk.tokenize import sent_tokenize,word_tokenize
```

```
nltk.download('punkt')
medical_corpus=[
    "The patient was diagnosed with hypertension after several visits
to the clinic.",
    "In clinical trials, new medications are often tested for their
efficacy and side effects.",
    "The physician recommended a follow-up appointment to monitor the
patient’s recovery.",
    "A balanced diet and regular exercise are essential for maintaining
cardiovascular health.",
    "The nurse administered the vaccine to the patient as part of the
immunization program.",
    "MRI scans are frequently used to assess the condition of the brain
and spinal cord.",
    "Chronic conditions, such as diabetes, require ongoing management
to prevent complications."
]
tokenized_corpus=[word_tokenize(sentence.lower()) for sentence in
medical_corpus]
print(tokenized_corpus)
print("Training Word2Vec model...")
model=Word2Vec(sentences=tokenized_corpus,vector_size=100,window=5,min_
count=1,workers=4,epochs=50)
print("Model Training Complete!")
words=list(model.wv.index_to_key)
embeddings=np.array([model.wv[word] for word in words])
tsne=TSNE(n_components=2,random_state=42,perplexity=5,n_iter=300)
tsne_result=tsne.fit_transform(embeddings)
plt.figure(figsize=(10,8))
plt.scatter(tsne_result[:,0],tsne_result[:,1],color="blue")
for i,word in enumerate(words):

plt.text(tsne_result[i,0]+0.02,tsne_result[i,1]+0.02,word,fontsize=12)
plt.title("Word Embeddings Visualization")
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.grid()
plt.show
def find_similar_words(input_word,top_n=5):
    try:
        similar_words=model.wv.most_similar(input_word,topn=top_n)
        print(f"Words similar to '{input_word}':")
        for word,similarity in similar_words:
```

```
            print(f"{word}({similarity:.2f})")
    except KeyError:
        print(f"'{input_word}'not found in vocabulary.")
find_similar_words("vaccine")
```

Pgm4:

```
!pip install groq
!pip install gensim
import groq
from gensim.models import KeyedVectors
import numpy as np
import os
# Get the vector for the word "king"
king_vector = model['king']

# Display the vector
print(king_vector)
from kaggle_secrets import UserSecretsClient
def generate_response(prompt,model_name="llama-3.3-70b-versatile"):
    user_secrets=UserSecretsClient()
    groq_api_key=user_secrets.get_secret("GROQ_API_KEY")
    if not groq_api_key:
        raise ValueError("GROQ_API_KEY environment variable is not
set.")
    client=groq.Client(api_key=groq_api_key)
    response=client.chat.completions.create(
        model=model_name,
        messages=[{"role":"system","content":"You are a helpul AI
assistant."},
                  {"role":"user","content":prompt}]
    )
    return response.choices[0].message.content
original_prompt="What is the scope of computer science at present."
response=generate_response(original_prompt)
print(response)
def enrich_prompt(prompt,model,max_enrichments=2):
    words=prompt.split()
    enriched_words=[]
    for word in words:
```

```
similar_words=get_similar_words(word,model,top_n=max_enrichments)
        filtered_similar_words=[w for w in similar_words if
w.isalpha()]
        if filtered_similar_words:
            enriched_words.append(word +
"("+",".join(filtered_similar_words)+")")
        else:
            enriched_words.append(word)
    return " ".join(enriched_words)
enriched_prompt=enrich_prompt(original_prompt,model)
print("original prompt:",original_prompt)
print("enriched prompt:",enriched_prompt)
original_response=generate_response(original_prompt)
enriched_response=generate_response(enriched_prompt)
print("\nOriginal Response:\n",original_response)
print("\nEnriched Response:\n",enriched_response)
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
def analyze_responses(original_response,enriched_response):
    vectorizer=TfidfVectorizer()
    tfidf_matrix=vectorizer.fit_transform([original_response,
                                           enriched_response])
    similarity_score=cosine_similarity(tfidf_matrix[0:1],
                                       tfidf_matrix[1:2])[0][0]
    original_length=len(original_response.split())
    enriched_length=len(enriched_response.split())
    print("\n Response Analysis:")
    print("Similarity score:",round(similarity_score,4))
    print("original response word count:",original_length)
    print("enriched response word count:",enriched_length)
analyze_responses(original_response,enriched_response)
```

Pgm5:

```
!pip install gensim scipy nltk
from gensim.models import KeyedVectors
import numpy as np
import os
import nltk
from nltk.corpus import wordnet
from nltk.tokenize import sent_tokenize
```

```
nltk.download('wordnet')
nltk.download('punkt')
def load_word_vectors():

model_path='/kaggle/input/google-word2vec/GoogleNews-vectors-negative300.bin'
    model=KeyedVectors.load_word2vec_format(model_path,binary=True)
    return model
model=load_word_vectors()
def get_similar_words(word,model,top_n=5):
    try:
        similar_words=model.most_similar(word,topn=top_n)
        return [w[0] for w in similar_words]
    except KeyError:
        return []
def get_synonyms(word):
    try:
        synonyms=set()
        for syn in wordnet.synsets(word):
            for lemma in syn.lemmas():
                synonyms.add(lemma.name())
        return list(synonyms)[:5]
    except LookupError:
        return []
def generate_story(seed_word,model):
    similar_words=get_similar_words(seed_word,model,top_n=3)
    synonyms=get_synonyms(seed_word)
    word_choices=list(set(similar_words + synonyms))
    while len(word_choices)<5:
        word_choices.append(seed_word)
    story_template=(
        f"Once upon a time, in a mystical land, there was an ancient {seed_word}."
        f"Legends spoke of its power hidden within the {word_choices[0]}."
        f"One evening, under a {word_choices[1]} sky, a young explorer named Alex set out on a journey."
        f"Guided by an old {word_choices[2]}, they discovered a secret passage leading to a hidden realm."
        f"Inside, they found an inscription written in {word_choices[3]} and uncovered the secret of {word_choices[4]}."
        f"This adventure would change their fate forever, unlocking mysteries long forgotten."
```

```
    )
    return " ".join(sent_tokenize(story_template))
seed_word="adventure"
generated_story=generate_story(seed_word,model)
print("Generated Story:")
print(generated_story)
```

arrr.pdf