```
from google.colab import drive
drive.mount('/content/drive')
!cp /content/drive/MyDrive/HCIb25-26/data/*.pkl .
```

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

```
#!wget https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.vec.gz
#!wget https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.uk.300.vec.gz
```

```
#!gunzip cc.en.300.vec.gz
#!gunzip cc.uk.300.vec.gz
```

```
#!wget https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.uk.300.bin.gz
#!wget https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.bin.gz
```

```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.4.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim) (2.0.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.16.3)
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (7.4.1)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart_open>=1.8.1->gensim) (2.0.0)
```

```
from gensim.models import KeyedVectors
import pandas as pd
import pickle
import numpy as np
```

```
#lang1_embeddings = KeyedVectors.load_word2vec_format("cc.en.300.vec", binary=False)

# Unpickling from a file
with open('en_vec.pkl', 'rb') as f:
    lang1_embeddings = pickle.load(f)
```

```
# 1 - en, 2 - uk
#lang2_embeddings = KeyedVectors.load_word2vec_format("cc.uk.300.vec", binary=False)

# Unpickling from a file
with open('uk_vec.pkl', 'rb') as f:
    lang2_embeddings = pickle.load(f)
```

```
"""

# Pickling to a file
with open('en_vec.pkl', 'wb') as f:
    pickle.dump(lang1_embeddings, f)

# Pickling to a file
with open('uk_vec.pkl', 'wb') as f:
    pickle.dump(lang2_embeddings, f)
"""
```

```
'\n\n# Pickling to a file\nwith open('en_vec.pkl', 'wb') as f:\n    pickle.dump(lang1_embeddings, f)\n\n# Pickling to a file
\nwith open('uk_vec.pkl', 'wb') as f:\n    pickle.dump(lang2_embeddings, f)\n'
```

```
#!pip install fasttext
```

```
#import fasttext.util
```

```
#fasttext.util.download_model('uk', if_exists='ignore')  # Ukrainian
#fasttext.util.download_model('en', if_exists='ignore')  # English
```

```
# 1 - en, 2 - uk

#lang1_embeddings = fasttext.load_model('cc.en.300.bin')
#lang2_embeddings = fasttext.load_model('cc.uk.300.bin')
```

```
!wget https://dl.fbaipublicfiles.com/arrival/dictionaries/en-uk.0-5000.txt
!wget https://dl.fbaipublicfiles.com/arrival/dictionaries/en-uk.5000-6500.txt
```

```
--2025-11-06 07:52:18--  https://dl.fbaipublicfiles.com/arrival/dictionaries/en-uk.0-5000.txt
Resolving dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)... 54.240.184.75, 54.240.184.92, 54.240.184.91, ...
Connecting to dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)|54.240.184.75|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 232454 (227K) [text/plain]
Saving to: ‘en-uk.0-5000.txt.1’

en-uk.0-5000.txt.1  100%[===================>] 227.01K  --.-KB/s    in 0.03s

2025-11-06 07:52:18 (6.50 MB/s) - ‘en-uk.0-5000.txt.1’ saved [232454/232454]

--2025-11-06 07:52:18--  https://dl.fbaipublicfiles.com/arrival/dictionaries/en-uk.5000-6500.txt
Resolving dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)... 54.240.184.75, 54.240.184.92, 54.240.184.91, ...
Connecting to dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)|54.240.184.75|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 51769 (51K) [text/plain]
Saving to: ‘en-uk.5000-6500.txt.1’

en-uk.5000-6500.txt 100%[===================>]  50.56K  --.-KB/s    in 0.01s

2025-11-06 07:52:18 (4.07 MB/s) - ‘en-uk.5000-6500.txt.1’ saved [51769/51769]
```

```python
def get_dict(file_name):
    my_file = pd.read_csv(file_name, delimiter='\t')
    etof = {}
    for i in range(len(my_file)):
        lang1 = my_file.loc[i][0]
        lang2 = my_file.loc[i][1]
        etof[lang1] = lang2

    return etof
```

```python
l1_l2_train = get_dict('en-uk.0-5000.txt')
l1_l2_test = get_dict('en-uk.5000-6500.txt')
```

```
/tmp/ipython-input-2844298481.py:5: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future ve
  lang1 = my_file.loc[i][0]
/tmp/ipython-input-2844298481.py:6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future ve
  lang2 = my_file.loc[i][1]
/tmp/ipython-input-2844298481.py:5: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future ve
```

```
  lang1 = my_file.loc[i][0]
/tmp/ipython-input-2844298481.py:6: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future ve
  lang2 = my_file.loc[i][1]
```

```
def get_matrices(l1_l2, l2_embeddings, l1_embeddings):
    """
    Вхід:
        l1_l2: словник перекладів з мови-1 на мову-2
        l2_embeddings: KeyedVectors векторів слів мови-2
        l1_embeddings: KeyedVectors векторів слів мови-1
    Вихід:
        X: матриця, де кожен рядок є вектором слова мови-1
        Y: матриця, де кожен рядок є вектором перекладу цього слова мовою-2
    """
    X_l = list()
    Y_l = list()

    l1_set = set(l1_embeddings.index_to_key)
    l2_set = set(l2_embeddings.index_to_key)

    for l1_word, l2_word in l1_l2.items():
        if l1_word in l1_set and l2_word in l2_set:
            X_l.append(l1_embeddings[l1_word])
            Y_l.append(l2_embeddings[l2_word])

    X = np.asarray(X_l)
    Y = np.asarray(Y_l)
    return X, Y
```

```
X_train, Y_train = get_matrices(l1_l2_train, lang2_embeddings, lang1_embeddings)
```

```
Y_train.shape
```

```
(4666, 300)
```

```
def compute_loss(X, Y, R):
    """
    Вхід:
        X: матриця векторів слів мови-1 (розмірність m x n)
        Y: матриця векторів слів мови-2 (розмірність m x n)
        R: матриця перетворення (розмірність n x n)
    Вихід:
        loss: значення функції втрат
    """
    m = X.shape[0]
    diff = np.dot(X, R) - Y
    loss = np.sum(diff**2) / m
    return loss
```

```
def compute_gradient(X, Y, R):
    """
    Вхід:
        X: матриця векторів слів мови-1 (розмірність m x n)
        Y: матриця векторів слів мови-2 (розмірність m x n)
        R: матриця перетворення (розмірність n x n)
    Вихід:
        gradient: матриця градієнту функції втрат
    """
    m = X.shape[0]
    gradient = np.dot(X.T, np.dot(X, R) - Y) * (2 / m)
    return gradient
```

```
def align_embeddings(X, Y, train_steps=100, learning_rate=0.0003, verbose=True,
                     compute_loss=compute_loss, compute_gradient=compute_gradient):
    """
    Вхід:
        X: матриця (розмірність m x n), кожен рядок - вектор слова мови-1
        Y: матриця (розмірність m x n), кожен рядок - вектор перекладу мовою-2
        train_steps: кількість ітерацій градієнтного спуску
        learning_rate: швидкість навчання
    Вихід:
        R: матриця перетворення
    """
    np.random.seed(129)
    m = X.shape[0]
    n = X.shape[1]
    R = np.random.rand(n, n)
    R = np.random.rand(n, n)

    for i in range(train_steps):
        if verbose and i % 10 == 0:
            print(f"Loss at iteration {i} is: {compute_loss(X, Y, R):.4f}")
        gradient = compute_gradient(X, Y, R)
        R -= learning_rate * gradient

    return R
```

```
R_train = align_embeddings(X_train, Y_train, train_steps=400, learning_rate=0.8)
```

```
Loss at iteration 0 is: 116.1960
Loss at iteration 10 is: 89.2659
Loss at iteration 20 is: 74.5666
Loss at iteration 30 is: 63.5847
Loss at iteration 40 is: 54.9662
Loss at iteration 50 is: 48.0138
Loss at iteration 60 is: 42.2934
Loss at iteration 70 is: 37.5143
Loss at iteration 80 is: 33.4727
Loss at iteration 90 is: 30.0209
Loss at iteration 100 is: 27.0481
Loss at iteration 110 is: 24.4699
Loss at iteration 120 is: 22.2202
Loss at iteration 130 is: 20.2466
Loss at iteration 140 is: 18.5070
Loss at iteration 150 is: 16.9671
Loss at iteration 160 is: 15.5987
Loss at iteration 170 is: 14.3783
Loss at iteration 180 is: 13.2865
Loss at iteration 190 is: 12.3066
Loss at iteration 200 is: 11.4248
Loss at iteration 210 is: 10.6292
Loss at iteration 220 is: 9.9095
Loss at iteration 230 is: 9.2570
Loss at iteration 240 is: 8.6640
Loss at iteration 250 is: 8.1241
Loss at iteration 260 is: 7.6315
Loss at iteration 270 is: 7.1813
Loss at iteration 280 is: 6.7689
Loss at iteration 290 is: 6.3906
Loss at iteration 300 is: 6.0430
Loss at iteration 310 is: 5.7231
Loss at iteration 320 is: 5.4282
Loss at iteration 330 is: 5.1559
Loss at iteration 340 is: 4.9043
Loss at iteration 350 is: 4.6713
Loss at iteration 360 is: 4.4553
Loss at iteration 370 is: 4.2548
Loss at iteration 380 is: 4.0685
Loss at iteration 390 is: 3.8951
```

```
def realign_embeddings(X, Y, R, train_steps=100, learning_rate=0.0003, verbose=True,
                     compute_loss=compute_loss, compute_gradient=compute_gradient):
    """
    Вхід:
        X: матриця (розмірність m x n), кожен рядок - вектор слова мови-1
        Y: матриця (розмірність m x n), кожен рядок - вектор перекладу мовою-2
        R: матриця перетворення
        train_steps: кількість ітерацій градієнтного спуску
        learning_rate: швидкість навчання
    Вихід:
        R: матриця перетворення донавчена (fune tuned)
    """
    #np.random.seed(129)
    m = X.shape[0]
    n = X.shape[1]
    #R = np.random.rand(n, n)

    for i in range(train_steps):
        if verbose and i % 10 == 0:
            print(f"Loss at iteration {i} is: {compute_loss(X, Y, R):.4f}")
        gradient = compute_gradient(X, Y, R)
        R -= learning_rate * gradient

    return R
```

```
R_train = realign_embeddings(X_train, Y_train, R_train, train_steps=100, learning_rate=0.8)
```

```
Loss at iteration 0 is: 1.6023
Loss at iteration 10 is: 1.5741
Loss at iteration 20 is: 1.5471
Loss at iteration 30 is: 1.5213
Loss at iteration 40 is: 1.4966
Loss at iteration 50 is: 1.4730
Loss at iteration 60 is: 1.4503
Loss at iteration 70 is: 1.4285
Loss at iteration 80 is: 1.4077
Loss at iteration 90 is: 1.3876
```

```
R_train = realign_embeddings(X_train, Y_train, R_train, train_steps=500, learning_rate=0.8)
```

```
Loss at iteration 0 is: 1.3684
Loss at iteration 10 is: 1.3499
Loss at iteration 20 is: 1.3321
Loss at iteration 30 is: 1.3150
Loss at iteration 40 is: 1.2985
Loss at iteration 50 is: 1.2827
Loss at iteration 60 is: 1.2674
Loss at iteration 70 is: 1.2527
Loss at iteration 80 is: 1.2385
Loss at iteration 90 is: 1.2248
Loss at iteration 100 is: 1.2116
Loss at iteration 110 is: 1.1989
Loss at iteration 120 is: 1.1865
Loss at iteration 130 is: 1.1746
Loss at iteration 140 is: 1.1631
Loss at iteration 150 is: 1.1520
Loss at iteration 160 is: 1.1413
Loss at iteration 170 is: 1.1308
Loss at iteration 180 is: 1.1208
Loss at iteration 190 is: 1.1110
Loss at iteration 200 is: 1.1015
Loss at iteration 210 is: 1.0924
Loss at iteration 220 is: 1.0835
Loss at iteration 230 is: 1.0749
Loss at iteration 240 is: 1.0665
Loss at iteration 250 is: 1.0584
Loss at iteration 260 is: 1.0505
Loss at iteration 270 is: 1.0428
Loss at iteration 280 is: 1.0354
Loss at iteration 290 is: 1.0282
Loss at iteration 300 is: 1.0212
Loss at iteration 310 is: 1.0143
Loss at iteration 320 is: 1.0077
Loss at iteration 330 is: 1.0012
Loss at iteration 340 is: 0.9949
Loss at iteration 350 is: 0.9888
Loss at iteration 360 is: 0.9828
Loss at iteration 370 is: 0.9770
Loss at iteration 380 is: 0.9714
Loss at iteration 390 is: 0.9659
Loss at iteration 400 is: 0.9605
Loss at iteration 410 is: 0.9552
Loss at iteration 420 is: 0.9501
Loss at iteration 430 is: 0.9451
Loss at iteration 440 is: 0.9403
Loss at iteration 450 is: 0.9355
Loss at iteration 460 is: 0.9309
Loss at iteration 470 is: 0.9264
Loss at iteration 480 is: 0.9219
Loss at iteration 490 is: 0.9176
```

```python
def cosine_similarity(A, B):
    cos = -10
    normb = np.linalg.norm(B, axis=1) # Calculate norm of each row in B

    if len(A.shape) == 1: # If A is a single vector
        norma = np.linalg.norm(A)
        dot = np.dot(A, B.T) # Calculate dot product with transpose of B
        epsilon = 1.0e-9 # to avoid division by 0
        cos = dot / (norma * normb + epsilon)
    else: # If A is a matrix
        norma = np.linalg.norm(A, axis=1)
        dot = np.dot(A, B.T) # Calculate dot product with transpose of B
        epsilon = 1.0e-9 # to avoid division by 0
        cos = dot / (norma * normb + epsilon)

    return cos
```

```python
def translate(word, R, l1_embeddings, l2_embeddings, k=5):
    """
    Вхід:
        word: слово мови-1 для перекладу
        R: матриця перетворення
        l1_embeddings: KeyedVectors векторів слів мови-1
        l2_embeddings: KeyedVectors векторів слів мови-2
        k: кількість найближчих сусідів для пошуку
    Вихід:
        переклад слова мовою-2
    """
    if word not in l1_embeddings.key_to_index:
        return "Слово відсутнє у словнику"

    l1_emb = l1_embeddings[word]
    l2_emb = np.dot(l1_emb, R)
    distances = cosine_similarity(l2_emb, l2_embeddings.vectors)
    top_k = distances.argsort()[-k:][::-1]

    return l2_embeddings.index_to_key[top_k[0]]
```

```
R_train = realign_embeddings(X_train, Y_train, R_train, train_steps=500, learning_rate=2.0)
```

```
Loss at iteration 0 is: 0.7640
Loss at iteration 10 is: 0.7607
Loss at iteration 20 is: 0.7575
Loss at iteration 30 is: 0.7544
Loss at iteration 40 is: 0.7515
Loss at iteration 50 is: 0.7486
Loss at iteration 60 is: 0.7459
Loss at iteration 70 is: 0.7432
Loss at iteration 80 is: 0.7406
Loss at iteration 90 is: 0.7381
Loss at iteration 100 is: 0.7357
Loss at iteration 110 is: 0.7333
Loss at iteration 120 is: 0.7311
Loss at iteration 130 is: 0.7289
Loss at iteration 140 is: 0.7267
Loss at iteration 150 is: 0.7247
Loss at iteration 160 is: 0.7226
Loss at iteration 170 is: 0.7207
Loss at iteration 180 is: 0.7188
Loss at iteration 190 is: 0.7170
Loss at iteration 200 is: 0.7152
Loss at iteration 210 is: 0.7134
Loss at iteration 220 is: 0.7118
Loss at iteration 230 is: 0.7101
Loss at iteration 240 is: 0.7085
Loss at iteration 250 is: 0.7070
Loss at iteration 260 is: 0.7054
Loss at iteration 270 is: 0.7040
Loss at iteration 280 is: 0.7025
Loss at iteration 290 is: 0.7011
Loss at iteration 300 is: 0.6998
Loss at iteration 310 is: 0.6985
Loss at iteration 320 is: 0.6972
Loss at iteration 330 is: 0.6959
Loss at iteration 340 is: 0.6947
Loss at iteration 350 is: 0.6935
Loss at iteration 360 is: 0.6923
Loss at iteration 370 is: 0.6912
Loss at iteration 380 is: 0.6901
Loss at iteration 390 is: 0.6890
Loss at iteration 400 is: 0.6879
Loss at iteration 410 is: 0.6869
Loss at iteration 420 is: 0.6859
Loss at iteration 430 is: 0.6849
Loss at iteration 440 is: 0.6839
Loss at iteration 450 is: 0.6830
Loss at iteration 460 is: 0.6821
Loss at iteration 470 is: 0.6812
Loss at iteration 480 is: 0.6803
Loss at iteration 490 is: 0.6794
```

```
def align_embeddings_loss(X, Y, train_steps=100, learning_rate=0.0003, epsilon = 0.1, verbose=True,
                     compute_loss=compute_loss, compute_gradient=compute_gradient):
    """
    Вхід:
        X: матриця (розмірність m x n), кожен рядок - вектор слова мови-1
        Y: матриця (розмірність m x n), кожен рядок - вектор перекладу мовою-2
        train_steps: кількість ітерацій градієнтного спуску
        learning_rate: швидкість навчання
    Вихід:
        R: матриця перетворення
    """
    np.random.seed(129)
    m = X.shape[0]
    n = X.shape[1]
    R = np.random.rand(n, n)

    for i in range(train_steps):
        if verbose and i % 10 == 0:
            eps = compute_loss(X, Y, R)
            if eps < epsilon:
              break
            print(f"Loss at iteration {i} is: {eps:.4f}")
        gradient = compute_gradient(X, Y, R)
        R -= learning_rate * gradient

    return R
```

```
R_train = align_embeddings_loss(X_train, Y_train, train_steps=4000, learning_rate=5.0, epsilon = 0.6312)
```

```
Loss at iteration 2500 is: 0.6312
Loss at iteration 2590 is: 0.6313
Loss at iteration 2600 is: 0.6312
Loss at iteration 2610 is: 0.6312
Loss at iteration 2620 is: 0.6312
Loss at iteration 2630 is: 0.6312
Loss at iteration 2640 is: 0.6312
Loss at iteration 2650 is: 0.6312
Loss at iteration 2660 is: 0.6312
Loss at iteration 2670 is: 0.6312
Loss at iteration 2680 is: 0.6312
Loss at iteration 2690 is: 0.6312
Loss at iteration 2700 is: 0.6312
Loss at iteration 2710 is: 0.6312
Loss at iteration 2720 is: 0.6312
Loss at iteration 2730 is: 0.6312
Loss at iteration 2740 is: 0.6312
Loss at iteration 2750 is: 0.6312
Loss at iteration 2760 is: 0.6312
Loss at iteration 2770 is: 0.6312
Loss at iteration 2780 is: 0.6312
Loss at iteration 2790 is: 0.6312
Loss at iteration 2800 is: 0.6312
```

```
translate("ukrainian", R_train, lang1_embeddings, lang2_embeddings)
```

```
'українська'
```

NER - Name Entity Recognition

```
phrase = """
first page article made discussion
"""
phrase_list = [line.strip() for line in phrase.strip().split(' ')]

result = ""
for word in phrase_list:
  result = result + " " + translate(word, R_train, lang1_embeddings, lang2_embeddings)
print("Переклад: ", result)
```

```
Переклад:   перший сторінку інтернет-блог зроблено обговорювання
```

```
phrase_list
```

```
['Melissa is now passing through the Bahamas as a Category 1 hurricane after making landfall in Cuba this morning']
```

```
def test_vocabulary(X, Y, R, l1_embeddings, l2_embeddings):
    """
    Вхід:
        X: матриця векторів слів мови-1
        Y: матриця векторів слів мови-2
        R: матриця перетворення
        l1_embeddings: KeyedVectors векторів слів мови-1
        l2_embeddings: KeyedVectors векторів слів мови-2
    Вихід:
        accuracy: точність перекладу
    """
    l1_words = list(l1_embeddings.index_to_key)
    l2_words = list(l2_embeddings.index_to_key)

    pred = np.dot(X, R)
    correct = 0
    for i in range(len(X)):
        l1_word = l1_words[i]
        pred_l2 = translate(l1_word, R, l1_embeddings, l2_embeddings, k=1)
        true_l2 = l2_words[i]
        if pred_l2 == true_l2:
            correct += 1

    accuracy = correct / len(X)
    return accuracy
```

```
X_test, Y_test = get_matrices(l1_l2_test, lang2_embeddings, lang1_embeddings)
```

```
X_test.shape
```

```
(1286, 300)
```

```
accuracy_test = test_vocabulary(X_test, Y_test, R_train, lang1_embeddings, lang2_embeddings)
print(f"Точність перекладу на тестовому наборі: {accuracy_test:.3f}")
```

```
import nltk
nltk.download('twitter_samples')
```

```
[nltk_data] Downloading package twitter_samples to /root/nltk_data...
[nltk_data]   Unzipping corpora/twitter_samples.zip.
True
```

```
from nltk.corpus import twitter_samples

all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
all_tweets = all_positive_tweets + all_negative_tweets
```

```
len(all_tweets)
```

```
10000
```

```
all_tweets[0]
```

```
'#FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this week :)'
```

```
import re, string
```

```
from nltk.corpus import stopwords
from nltk.tokenize import TweetTokenizer
from nltk.stem import PorterStemmer
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
def process_tweet(tweet):
    '''
    Input:
        tweet: a string containing a tweet
    Output:
        tweets_clean: a list of words containing the processed tweet

    '''
    stemmer = PorterStemmer()
    stopwords_english = stopwords.words('english')
    # remove stock market tickers like $GE
    tweet = re.sub(r'\$\w*', '', tweet)
    # remove old style retweet text "RT"
    tweet = re.sub(r'^RT[\s]+', '', tweet)
    # remove hyperlinks
    tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
    # remove hashtags
    # only removing the hash # sign from the word
    tweet = re.sub(r'#', '', tweet)
    # tokenize tweets
    tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True,
                               reduce_len=True)
    tweet_tokens = tokenizer.tokenize(tweet)

    tweets_clean = []
    for word in tweet_tokens:
        if (word not in stopwords_english and  # remove stopwords
            word not in string.punctuation):  # remove punctuation
            # tweets_clean.append(word)
            stem_word = stemmer.stem(word)  # stemming word
            tweets_clean.append(stem_word)

    return tweets_clean
```

```
def get_tweet_embedding(tweet, l1_embeddings):
    """
    Вхід:
        tweet: рядок з текстом твіта
        l1_embeddings: KeyedVectors векторів слів
    Вихід:
        tweet_embedding: вектор ознак для твіта
    """
    words = process_tweet(tweet)
    embeddings = [l1_embeddings[w] if w in l1_embeddings.key_to_index else np.zeros(300) for w in words]
    if not embeddings:  # Handle empty embeddings list
        return np.zeros(300)
    tweet_embedding = np.mean(embeddings, axis=0)
    return tweet_embedding

def get_all_tweet_embeddings(tweets, l1_embeddings):
    """
    Вхід:
        tweets: список твітів
        l1_embeddings: словник векторів слів
    Вихід:
        tweet_embeddings: словник {id_твіта: вектор_твіта}
        tweet_id_map: список id твітів
    """
    tweet_embeddings = {}
    tweet_id_map = []
    for i, tweet in enumerate(tweets):
        tweet_embeddings[i] = get_tweet_embedding(tweet, l1_embeddings)
        tweet_id_map.append(i)
    return tweet_embeddings, tweet_id_map
```

```
tweet_embeddings, tweet_id_map = get_all_tweet_embeddings(all_tweets, lang1_embeddings)
print(f"Кількість твітів: {len(tweet_id_map)}")
print(f"Розмірність векторів твітів: {tweet_embeddings[0].shape}")
```

```
Кількість твітів: 10000
Розмірність векторів твітів: (300,)
```

```
def hash_func(embedding, planes):
    """
    Вхід:
        embedding: вектор ознак (розмірність 1 x n)
        planes: матриця випадкових гіперплощин (розмірність n x n_planes)
    Вихід:
        hash_value: хеш-значення
    """
    # Ensure embedding is a 2D array for matrix multiplication
    embedding = embedding.reshape(1, -1)
    dots = np.dot(embedding, planes)
    # Get the sign of the dot product
    signs = np.sign(dots)
    # Convert signs to 0 or 1
    binary_signs = (signs > 0).astype(int)
    # Calculate the hash value
    hash_value = 0
    for i in range(binary_signs.shape[1]):
        hash_value += binary_signs[0, i] * (2 ** i)
    return hash_value
```

```
def make_hash_table(embeddings, planes):
    """
    Вхід:
        embeddings: набір векторів ознак
        planes: матриця випадкових гіперплощин
    Вихід:
        hash_table: хеш-таблиця у вигляді словника:
                    {хеш_значення: [список векторів з цим хеш_значенням]}
    """
    hash_table = {}
    for i, embedding in enumerate(embeddings):
        hash_value = hash_func(embedding, planes)
        hash_table.setdefault(hash_value, []).append(i)
    return hash_table
```

```
def init_lsh(embeddings, n_planes, n_tables):
    """
    Вхід:
        embeddings: набір векторів ознак
        n_planes: кількість гіперплощин
        n_tables: кількість хеш-таблиць
    Вихід:
        tables: список хеш-таблиць
        planes_list: список матриць випадкових гіперплощин
    """
    planes_list = [np.random.normal(size=(embeddings.shape[1], n_planes))
                   for _ in range(n_tables)]
    tables = [make_hash_table(embeddings, planes) for planes in planes_list]
```

```
        tables = [make_hash_table(embeddings, planes) for planes in planes_list]
        return tables, planes_list
```

```
tables, planes_list = init_lsh(np.array(list(tweet_embeddings.values())), n_planes=10, n_tables=10)
```

```
tables
```

```
  np.int64(76): [9152, 9815]
  np.int64(76): [9132, 9815],
  np.int64(917): [9224],
  np.int64(104): [9303],
  np.int64(973): [9317],
  np.int64(873): [9326],
  np.int64(283): [9340],
  np.int64(879): [9348],
  np.int64(438): [9406],
  np.int64(795): [9444],
  np.int64(492): [9518],
  np.int64(181): [9610],
  np.int64(197): [9667],
  np.int64(928): [9739],
  np.int64(787): [9822],
  np.int64(951): [9834],
  np.int64(860): [9845],
  np.int64(702): [9847],
  np.int64(824): [9850],
  np.int64(919): [9854],
  np.int64(808): [9916],
  np.int64(691): [9958],
  np.int64(903): [9978],
  np.int64(190): [9990]}]
```

```
len(planes_list)
```

```
10
```

```
planes_list[0].shape
```

```
(300, 10)
```

```
def lsh_knn(tweet_id, embedding, tables, planes_list, k=5):
    """
    Вхід:
        tweet_id: індекс твіта
        embedding: вектор ознак твіта
        tables: список хеш-таблиць
        planes_list: список матриць випадкових гіперплощин
        k: кількість найближчих сусідів
    Вихід:
        neighbors: список індексів k найближчих сусідів
    """
    candidates = set()
    for table, planes in zip(tables, planes_list):
        hash_value = hash_func(embedding, planes)
        candidates.update(table.get(hash_value, []))

    candidates.discard(tweet_id)
    candidates = list(candidates)
    candidate_embeddings = np.array([tweet_embeddings[i] for i in candidates])

    distances = cosine_similarity(embedding.reshape(1, -1), candidate_embeddings)
    top_k_idx = np.argsort(distances.squeeze())[-k:][::-1]
    neighbors = [candidates[i] for i in top_k_idx]

    return neighbors
```

```
tweet_id = 0 # чи якийсь інший номер або просто уведений користувачем
#tweet = all_tweets[tweet_id]
tweet = """
The Guardian has only ever published 15 zero-star reviews. Here they all are
As Kim Kardashian’s All’s Fair sets a new low for TV, we revisit every single thing our critics have mercilessly panned. Brac

Lucy Mangan’s Guardian review of Kim Kardashian’s new Disney+ legal drama All’s Fair was something of a rarity. Not necessari
"""
```

```
print(f"Запит: {tweet}")

embedding = get_tweet_embedding(tweet, lang1_embeddings)
```

```
Запит:
The Guardian has only ever published 15 zero-star reviews. Here they all are
As Kim Kardashian’s All’s Fair sets a new low for TV, we revisit every single thing our critics have mercilessly panned. Brace

Lucy Mangan’s Guardian review of Kim Kardashian’s new Disney+ legal drama All’s Fair was something of a rarity. Not necessaril
```

```
embedding
```

```
        1.13019228e-02,  2.77499964e-03, -3.87923080e-02,  6.12346142e-02,
        3.22423077e-02, -3.39615416e-03,  3.39442311e-02, -1.42884616e-03,
        5.39807717e-03, -1.11903845e-02,  3.45769272e-03, -1.32326917e-02,
       -3.22730773e-02,  5.34616142e-04, -2.78942313e-02,  2.85961580e-03,
        6.71500000e-02, -1.88269151e-03,  5.39115390e-02,  4.26884613e-02,
       -2.54057701e-02,  4.20538462e-02,  2.66538449e-03, -8.02884612e-03,
       -1.36153841e-02,  3.61192314e-02,  2.84173082e-02, -3.18999988e-02,
       -2.31173098e-02, -9.34999991e-03, -6.49615393e-03,  2.30903852e-02,
        2.29384616e-02,  3.61788458e-02, -2.35288465e-02,  8.00961595e-03,
        8.15961587e-03, -2.60596166e-02, -1.55461541e-02, -8.49423060e-03,
       -6.97057708e-02, -2.92788462e-02, -1.40192302e-02, -5.78653779e-03,
       -2.02346160e-02, -9.15961555e-03,  1.64480779e-02,  2.58576918e-02,
        9.65385019e-04, -3.03961549e-02,  9.85192343e-03,  2.24500005e-02,
       -4.89807696e-02, -5.32884596e-03, -5.25461518e-02,  4.18653822e-03,
        4.15826932e-02, -4.97057694e-02,  2.09057692e-02, -7.09769240e-02,
       -4.09403871e-02,  4.88115380e-02, -9.53846062e-04,  2.04173069e-02,
       -4.70346158e-02,  3.85019238e-02, -1.59019231e-02,  3.55884616e-02,
        3.84307692e-02,  3.31365378e-02, -2.60563461e-01, -1.10788456e-02,
       -1.02826922e-02, -3.80576896e-03, -1.02192310e-01, -6.35961499e-03,
       -2.17538461e-02, -1.11250011e-02, -1.70000004e-03, -2.94576931e-02,
        5.77788452e-02,  5.50634592e-02,  6.75249977e-02,  5.74230761e-02,
        4.13057690e-02,  7.07115299e-03,  6.37211534e-02, -1.86403846e-02,
        3.73596153e-02, -3.40442305e-02,  5.86153962e-03,  5.05769271e-03,
       -7.87884607e-03,  6.94423148e-03, -6.08461558e-03, -3.70730765e-02,
       -2.52942312e-02, -1.91480771e-02,  8.62115407e-03, -2.41769224e-02,
        1.02499980e-03, -1.01692309e-02,  1.97365383e-02,  5.88269254e-03,
       -2.45884612e-02,  4.69076928e-02, -3.57884588e-03, -1.73865392e-02,
        4.67499976e-03, -9.09038505e-03, -1.00942309e-02, -1.57153846e-02,
        2.01153842e-03,  3.43076926e-02,  1.19230728e-03,  3.57884590e-03,







        7.01538412e-03,  7.32865350e-02,  6.22019233e-02,  7.39423057e-03
       -7.01538412e-03,  7.32863350e-02, -6.22019233e-02,  7.39423057e-03,
        1.70403841e-02,  1.38442307e-02,  2.26076924e-02, -4.83653846e-03,
       -1.70442302e-02, -7.36192311e-02,  5.41653840e-02,  1.49846154e-02,
        8.61538539e-04,  1.36711539e-02,  6.88461583e-03, -1.48500007e-02,
       -9.61346147e-03,  6.65634611e-02,  2.93846226e-03, -1.93634621e-02,
       -3.08807698e-02,  2.43942314e-02, -1.10603843e-01, -4.25211546e-02,
        2.39884613e-02, -5.28653778e-03, -7.56923095e-03, -2.77115410e-03,
       -3.50230783e-02, -2.30730771e-02,  3.73403845e-02,  5.32691891e-04,
       -1.66288462e-01,  3.75961540e-02,  9.34038488e-03,  3.48057690e-02,
        1.32134616e-02,  1.21074999e-01,  7.02307778e-03,  1.64557696e-02])
```

```
neighbors = lsh_knn(tweet_id, embedding, tables, planes_list, k=3)

print("Найбільш схожі твіти:")
for neighbor_id in neighbors:
    print(all_tweets[neighbor_id])
```

```
Найбільш схожі твіти:
I’d really like to handle Murielle Ahouré’s PR/Brand Image. She’s  missing too many opportunities
 :(
. @Ben_Barker1989 yeah doesn’t appear to be working. But I want a free cornetto :( (not a strawberry one, they’re a fake corne
Rape time at it’s best :D http://t.co/fC5U1TusUk &gt;&gt; http://t.co/GBxQmizM7O http://t.co/0Um6xgpsAn
```

```
Start coding or generate with AI.
```