```
!pip install fasttext tqdm

Collecting fasttext
  Downloading fasttext-0.9.3.tar.gz (73 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/73.4 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 73.4/73.4 kB 2.1 MB/s eta
0:00:00
ents to build wheel ... etadata (pyproject.toml) ... ent already
satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (4.66.5)
Collecting pybind11>=2.2 (from fasttext)
  Using cached pybind11-2.13.6-py3-none-any.whl.metadata (9.5 kB)
Requirement already satisfied: setuptools>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from fasttext) (71.0.4)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from fasttext) (1.26.4)
Using cached pybind11-2.13.6-py3-none-any.whl (243 kB)
Building wheels for collected packages: fasttext
  Building wheel for fasttext (pyproject.toml) ... e=fasttext-0.9.3-
cp310-cp310-linux_x86_64.whl size=4296188
sha256=758126956cf26fec0e403767b244cf73c56c2d3c7163d20eda438c321402a54
e
  Stored in directory:
/root/.cache/pip/wheels/0d/a2/00/81db54d3e6a8199b829d58e02cec2ddb20ce3
e59fad8d3c92a
Successfully built fasttext
Installing collected packages: pybind11, fasttext
Successfully installed fasttext-0.9.3 pybind11-2.13.6

!wget https://dl.fbaipublicfiles.com/fasttext/vectors-
crawl/cc.en.300.bin.gz
!wget https://dl.fbaipublicfiles.com/fasttext/vectors-
crawl/cc.hi.300.bin.gz
!gunzip cc.en.300.bin.gz
!gunzip cc.hi.300.bin.gz

--2024-09-25 07:20:43--
https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.bin.gz
Resolving dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)...
13.35.7.38, 13.35.7.82, 13.35.7.128, ...
Connecting to dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)|
13.35.7.38|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4503593528 (4.2G) [application/octet-stream]
Saving to: ‘cc.en.300.bin.gz’

cc.en.300.bin.gz    100%[===================>]   4.19G   110MB/s    in
39s

2024-09-25 07:21:22 (111 MB/s) - ‘cc.en.300.bin.gz’ saved
[4503593528/4503593528]
```

```
--2024-09-25 07:21:22--
https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.hi.300.bin.gz
Resolving dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)...
13.35.7.38, 13.35.7.82, 13.35.7.128, ...
Connecting to dl.fbaipublicfiles.com (dl.fbaipublicfiles.com)|
13.35.7.38|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4371554972 (4.1G) [application/octet-stream]
Saving to: ‘cc.hi.300.bin.gz’

cc.hi.300.bin.gz    100%[===================>]   4.07G  23.8MB/s    in
38s

2024-09-25 07:22:00 (111 MB/s) - ‘cc.hi.300.bin.gz’ saved
[4371554972/4371554972]
```

```python
# Cross-Lingual Word Embedding Alignment Project

# Import necessary libraries
import numpy as np
import pandas as pd
import fasttext
import fasttext.util
from sklearn.metrics.pairwise import cosine_similarity
from scipy.linalg import orthogonal_procrustes
import matplotlib.pyplot as plt
from tqdm import tqdm

# Step 1: Data Preparation

def load_fasttext_embeddings(lang):
    """
    Load pre-trained FastText embeddings for a given language.
    """
    model = fasttext.load_model(f'cc.{lang}.300.bin')
    return model

def get_top_words(model, n=100000):
    """
    Get the top n most frequent words from the model.
    """
    words = []
    for word in model.get_words():
        words.append(word)
        if len(words) == n:
            break
    return words
```

```
def load_muse_dictionary(file_path):
    """
    Load the MUSE bilingual dictionary.
    """
    with open(file_path, 'r', encoding='utf-8') as f:
        return [line.strip().split() for line in f]

# Step 2: Embedding Alignment

def align_embeddings(src_emb, tgt_emb, src_words, tgt_words,
dictionary):
    """
    Align source embeddings to target embeddings using Procrustes
method.
    """
    src_indices = [src_words.index(pair[0]) for pair in dictionary if
pair[0] in src_words and pair[1] in tgt_words]
    tgt_indices = [tgt_words.index(pair[1]) for pair in dictionary if
pair[0] in src_words and pair[1] in tgt_words]

    src_aligned = src_emb[src_indices]
    tgt_aligned = tgt_emb[tgt_indices]

    R, _ = orthogonal_procrustes(src_aligned, tgt_aligned)
    src_emb_aligned = src_emb @ R

    return src_emb_aligned, R

# Step 3: Evaluation

def word_translation(src_word, src_emb_aligned, src_words, tgt_emb,
tgt_words, k=5):
    """
    Translate a source word to target language using aligned
embeddings.
    """
    if src_word not in src_words:
        return []

    src_index = src_words.index(src_word)
    src_vec = src_emb_aligned[src_index].reshape(1, -1)

    similarities = cosine_similarity(src_vec, tgt_emb)[0]
    top_indices = similarities.argsort()[-k:][::-1]

    return [tgt_words[i] for i in top_indices]

def evaluate_translation(test_dict, src_emb_aligned, src_words,
tgt_emb, tgt_words):
    """
```

```
    Evaluate translation accuracy using Precision@1 and Precision@5.
    """
    correct_1 = 0
    correct_5 = 0
    total = 0

    for src_word, tgt_word in tqdm(test_dict):
        if src_word in src_words and tgt_word in tgt_words:
            translations = word_translation(src_word, src_emb_aligned,
src_words, tgt_emb, tgt_words)
            if translations:
                if translations[0] == tgt_word:
                    correct_1 += 1
                if tgt_word in translations:
                    correct_5 += 1
                total += 1

    p1 = correct_1 / total if total > 0 else 0
    p5 = correct_5 / total if total > 0 else 0

    return p1, p5

def analyze_cosine_similarities(src_emb_aligned, tgt_emb, src_words,
tgt_words, pairs):
    """
    Analyze cosine similarities between word pairs.
    """
    similarities = []
    for src_word, tgt_word in pairs:
        if src_word in src_words and tgt_word in tgt_words:
            src_index = src_words.index(src_word)
            tgt_index = tgt_words.index(tgt_word)
            src_vec = src_emb_aligned[src_index].reshape(1, -1)
            tgt_vec = tgt_emb[tgt_index].reshape(1, -1)
            similarity = cosine_similarity(src_vec, tgt_vec)[0][0]
            similarities.append((src_word, tgt_word, similarity))
    return similarities

def ablation_study(src_emb, tgt_emb, src_words, tgt_words, train_dict,
test_dict, sizes):
    """
    Perform ablation study with different training dictionary sizes.
    """
    results = []
    for size in sizes:
        print(f"Training with {size} word pairs...")
        train_subset = train_dict[:size]
        src_emb_aligned, _ = align_embeddings(src_emb, tgt_emb,
src_words, tgt_words, train_subset)
        p1, p5 = evaluate_translation(test_dict, src_emb_aligned,
```

```
src_words, tgt_emb, tgt_words)
        results.append((size, p1, p5))
    return results

# Main execution

if __name__ == "__main__":
    # Load pre-trained FastText embeddings
    print("Loading FastText embeddings...")
    en_model = load_fasttext_embeddings('en')
    hi_model = load_fasttext_embeddings('hi')

    # Get top 100,000 words for each language
    print("Extracting top words...")
    en_words = get_top_words(en_model)
    hi_words = get_top_words(hi_model)

    # Extract embeddings
    en_emb = np.array([en_model.get_word_vector(w) for w in en_words])
    hi_emb = np.array([hi_model.get_word_vector(w) for w in hi_words])

    # Load MUSE dictionaries
    print("Loading MUSE dictionaries...")
    train_dict = load_muse_dictionary('MUSE/en-hi.0-5000.txt')
    test_dict = load_muse_dictionary('MUSE/en-hi.5000-6500.txt')

    # Align embeddings
    print("Aligning embeddings...")
    en_emb_aligned, R = align_embeddings(en_emb, hi_emb, en_words,
hi_words, train_dict)

    # Evaluate translation
    print("Evaluating translation...")
    p1, p5 = evaluate_translation(test_dict, en_emb_aligned, en_words,
hi_emb, hi_words)
    print(f"Precision@1: {p1:.4f}")
    print(f"Precision@5: {p5:.4f}")

    # Analyze cosine similarities
    print("Analyzing cosine similarities...")
    similarities = analyze_cosine_similarities(en_emb_aligned, hi_emb,
en_words, hi_words, test_dict[:100])
    for src, tgt, sim in similarities[:10]:
        print(f"{src} - {tgt}: {sim:.4f}")

    # Ablation study
    print("Performing ablation study...")
    sizes = [1000, 2000, 3000, 4000, 5000]
    ablation_results = ablation_study(en_emb, hi_emb, en_words,
hi_words, train_dict, test_dict, sizes)
```

```
    # Plot ablation study results
    plt.figure(figsize=(10, 6))
    sizes, p1_scores, p5_scores = zip(*ablation_results)
    plt.plot(sizes, p1_scores, marker='o', label='Precision@1')
    plt.plot(sizes, p5_scores, marker='o', label='Precision@5')
    plt.xlabel('Training Dictionary Size')
    plt.ylabel('Precision')
    plt.title('Impact of Training Dictionary Size on Translation
Accuracy')
    plt.legend()
    plt.grid(True)
    plt.savefig('ablation_study_results.png')
    plt.close()

    print("Project completed successfully!")
```