04 Task- Sentence Similarity using Sentence Transformers

Problem: Sentence Similarity using Sentence Transformers

You are tasked with calculating the similarity between a set of input sentences using a pre-trained Sentence Transformer model.

Hint: Use Hugging Face to get the model and Google Colab to run.

Problem Description

You are given a list of sentences. Your task is to compute the embeddings for each sentence using a pre-trained Sentence Transformer model and calculate the pairwise cosine similarity between all sentences in the list.

The cosine similarity between two vectors is a measure of similarity between them. It ranges from 0 to 1, where 1 means identical and 0 means completely dissimilar.

You need to implement the following steps:

1. **Load a Pre-trained Sentence Transformer Model:** Use a small, efficient Sentence Transformer model (e.g., `'all-MiniLM-L6-v2'`).
2. **Create Embeddings for Sentences:** Use the model to compute sentence embeddings for each sentence in the input list.
3. **Compute Pairwise Cosine Similarity:** Calculate the pairwise cosine similarity between all sentence embeddings. This will result in a similarity matrix where the element at index `[i][j]` is the cosine similarity between the `i`-th and `j`-th sentence.
4. **Return Sorted Similarities:** Return the similarities as a flat list of floating-point values (rounded to 4 decimal places), sorted in descending order.

Input Format

You will be given a list of sentences as follows:

```
sentences = [
    "The quick brown fox jumps over the lazy dog",
    "A fast brown fox leaps over the lazy dog",
    "The weather is sunny and pleasant today",
    "Cats are adorable pets"
]
```

Output Format

You need to return a sorted list of cosine similarity scores (rounded to 4 decimal places) between all pairs of sentences.

For example, given the sentences above, your output should look like:

```
[1.0000, 0.9172, 0.7001, 0.6791, 0.6545, 0.5432, 0.4993, 0.4742, 0.3445, 0.2113]
```

Constraints

- The input list `sentences` will contain at most 1000 sentences.
- Each sentence will have at most 500 characters.
- The similarity score will be a floating-point number between 0 and 1 (inclusive).

Hints

1. **Pre-trained Model**:
   - Use the `SentenceTransformer` class from the `sentence-transformers` library to load a pre-trained model. Example: `model = SentenceTransformer('all-MiniLM-L6-v2')`.
2. **Creating Embeddings**:
   - Use the `model.encode()` method to generate sentence embeddings for all the sentences.
3. **Cosine Similarity**:
   - Use `sklearn.metrics.pairwise.cosine_similarity` to compute the pairwise cosine similarity between all sentence embeddings. The output will be a similarity matrix.
4. **Flatten the Matrix**:
   - You can extract the upper triangle of the similarity matrix, excluding the diagonal (since it represents the similarity of a sentence with itself), and flatten the results into a list.
5. **Sorting**:
   - After computing the similarities, sort the list in descending order.
6. **Rounding**:
   - Round each similarity score to four decimal places before returning the list.

Example Walkthrough

Given the following sentences:

```
sentences = [
    "The quick brown fox jumps over the lazy dog",
    "A fast brown fox leaps over the lazy dog",
    "The weather is sunny and pleasant today",
    "Cats are adorable pets"
]
```

1. First, create embeddings for each sentence.
2. Compute the pairwise cosine similarity between all pairs of sentences.
3. Sort the similarity scores and return the result as a flat list.

The output for the above example would be:

```
[1.0000, 0.9172, 0.7001, 0.6791, 0.6545, 0.5432, 0.4993, 0.4742, 0.3445, 0.2113]
```

Function Signature

```
def compute_similarity(sentences: List[str]) -> List[float]:
    pass
```

Good luck!