CSI4001-NATURAL LANGUAGE PROCESSING

Document Similarity Analysis using BERT-based Embeddings

&

Advanced PDF Document Similarity Analysis using BERT-based Embeddings

Winter Semester(2024-2025)

E-record

Submitted by:

Name: CHAARULATHA J

Reg no: 22MID0317

Document Similarity Analysis using BERT-based Embeddings

Table of Contents:

1. AIM
2. INTRODUCTION
3. METHODOLOGY
4. IMPLEMENTATION
5. CODE SNIPPET
6. RESULTS AND ANALYSIS
7. APPLICATIONS
8. CONCLUSION
9. FUTURE ENHANCEMENTS
10. COMPARISON TABLE

1. AIM:

The aim of this project is to compute and analyze the similarity between two text documents using BERT-based sentence embeddings and cosine similarity. The approach ensures efficient and accurate similarity detection by leveraging pre-trained transformer models.

2. INTRODUCTION:

Document similarity analysis is a crucial task in various domains, such as plagiarism detection, information retrieval, and text clustering. Traditional approaches rely on lexical similarity methods, which may not capture the semantic meaning of the text. This project uses the Sentence-BERT model to generate embeddings and applies cosine similarity to measure the degree of similarity between two documents.

3. METHODOLOGY:

The project follows these steps to compute document similarity:

1. **Document Preprocessing:**
    - Read the input documents.
    - Convert text to lowercase for uniformity.
2. **Embedding Generation:**
    - Utilize the pre-trained BERT-based all-MiniLM-L6-v2 model from the sentence-transformers library to generate dense vector representations of the documents.
3. **Similarity Computation:**
    - Compute the cosine similarity between the two document embeddings.
    - Compare the similarity score against a predefined threshold (e.g., 0.7) to determine if the documents are similar.
    -

4. IMPLEMENTATION:

The implementation uses Python and the following libraries:

- sentence-transformers: For generating sentence embeddings.
- scikit-learn: For computing cosine similarity.
- numpy: For handling numerical computations (if needed).

5.CODE SNIPPET:

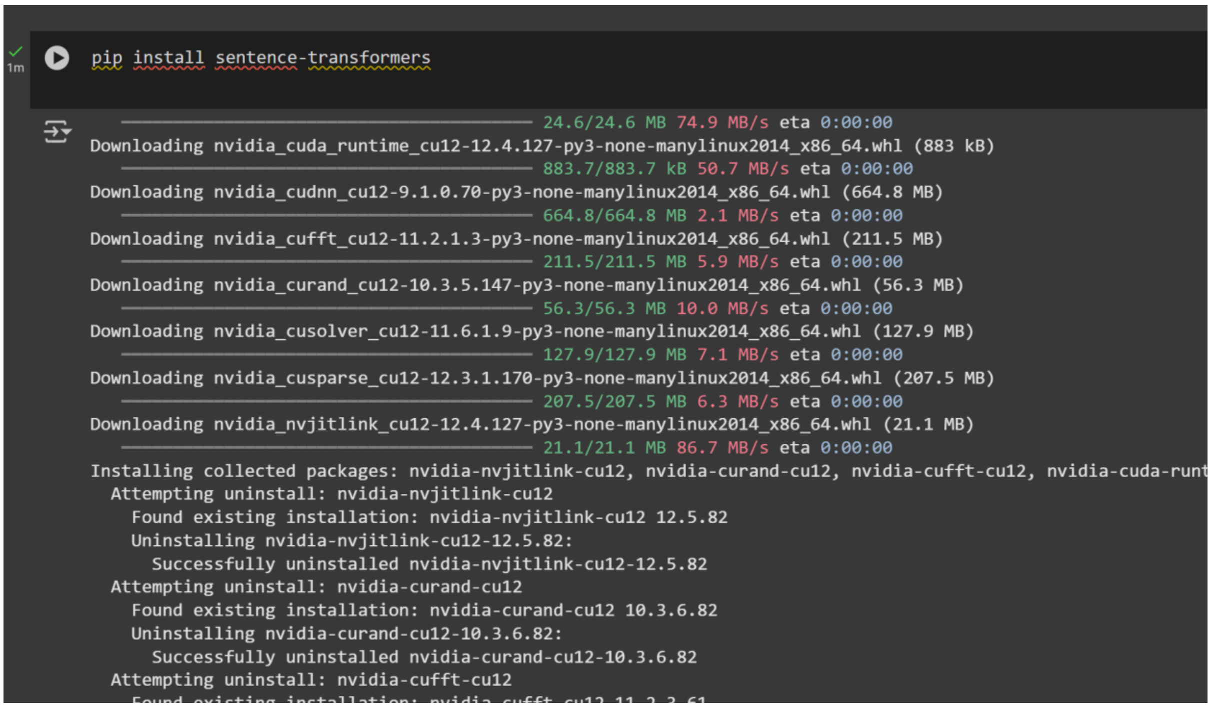
```
from sentence_transformers import SentenceTransformer

from sklearn.metrics.pairwise import cosine_similarity

def read_document(file_path):

    """Reads the content of a document."""

    try:

        with open(file_path, 'r', encoding='utf-8') as file:

            return file.read().strip().lower()  # Lowercasing for uniformity

    except FileNotFoundError:

        print(f"Error: File '{file_path}' not found.")

        return None

def check_document_similarity(file1, file2, threshold=0.7):

    """Computes cosine similarity between two text documents using BERT embeddings."""

    # Read documents

    doc1 = read_document(file1)

    doc2 = read_document(file2)

    if doc1 is None or doc2 is None:

        return

    # Load a pre-trained BERT-based sentence transformer model

    model = SentenceTransformer('all-MiniLM-L6-v2')

    # Generate embeddings for both documents

    embeddings = model.encode([doc1, doc2])

    # Compute cosine similarity

    similarity_score = cosine_similarity([embeddings[0]], [embeddings[1]])[0][0]

    # Print similarity score

    print(f"Cosine Similarity: {similarity_score:.4f}")

    # Check if documents are similar
    if similarity_score >= threshold:
        print("✅ Documents are similar.")
    else:
        print("❌ Documents are not similar.")
# Example usage
file1 = input("Enter the path of the first document: ")
file2 = input("Enter the path of the second document: ")
check_document_similarity(file1, file2)
```

6. RESULTS AND ANALYSIS:

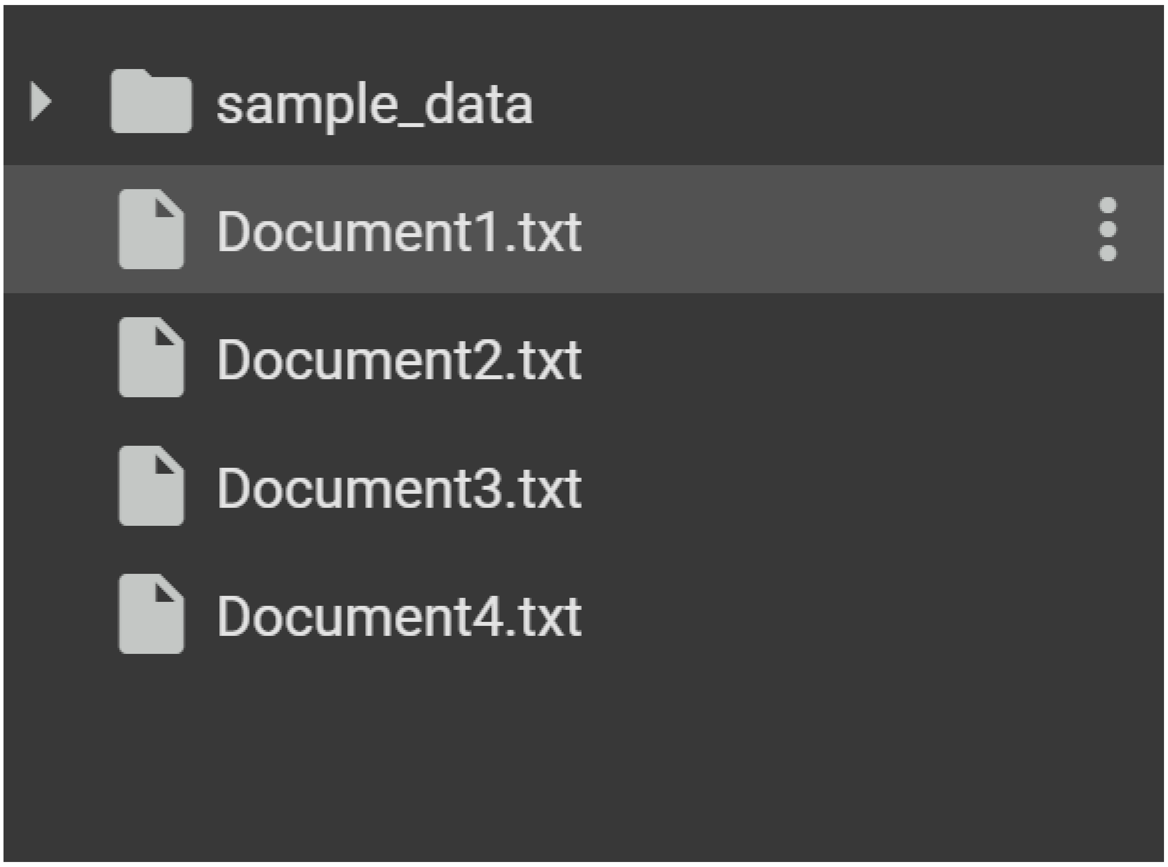
6.1 Experimental Results

The similarity analysis was conducted on four different document pairs

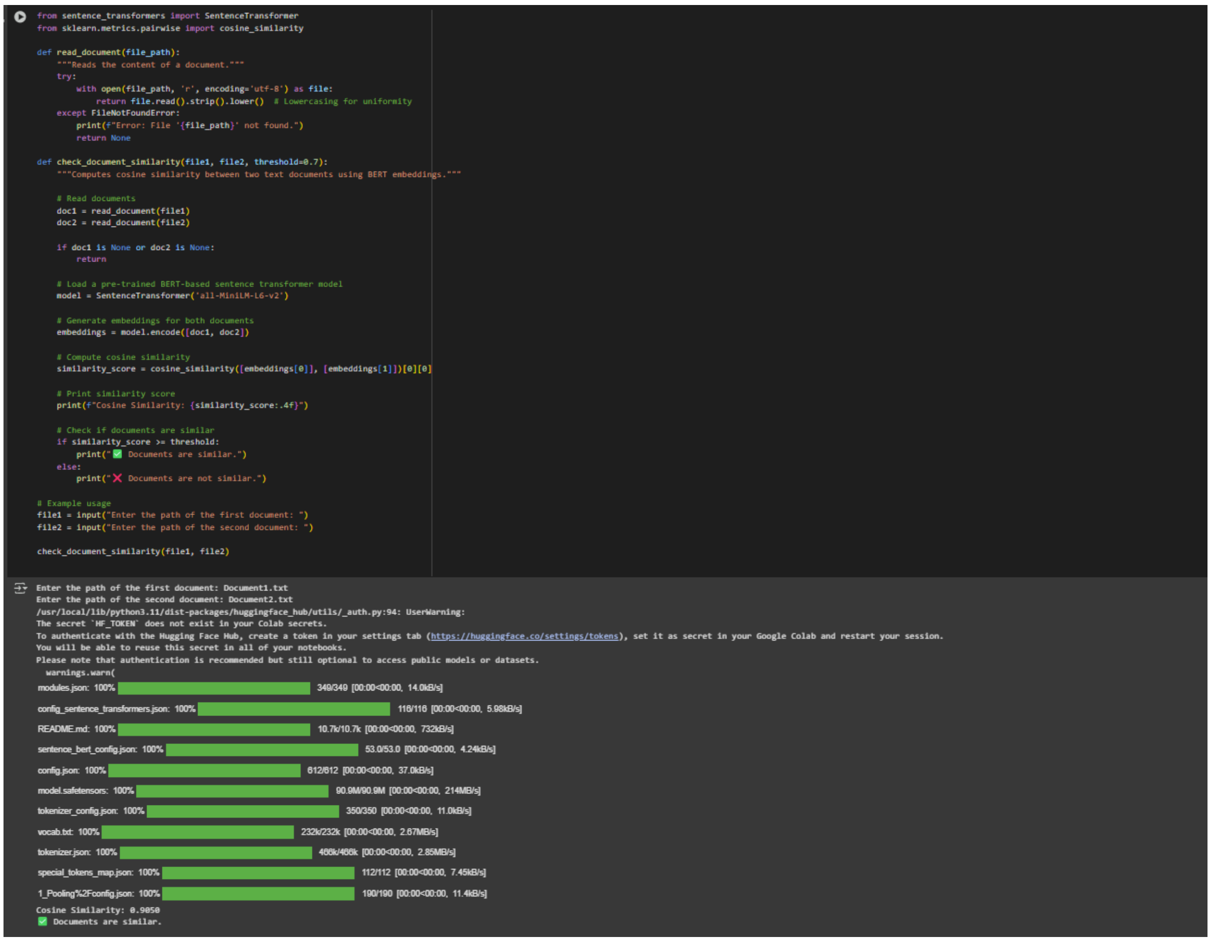
These results demonstrate that the BERT-based embeddings effectively capture the semantic meaning of text. The model correctly identifies similar and non-similar document pairs based on a threshold of 0.7.



```
pip install sentence-transformers

    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 74.9 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 50.7 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta 0:00:00
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.9 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 10.0 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.1 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.3 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 86.7 MB/s eta 0:00:00
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runt
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
```

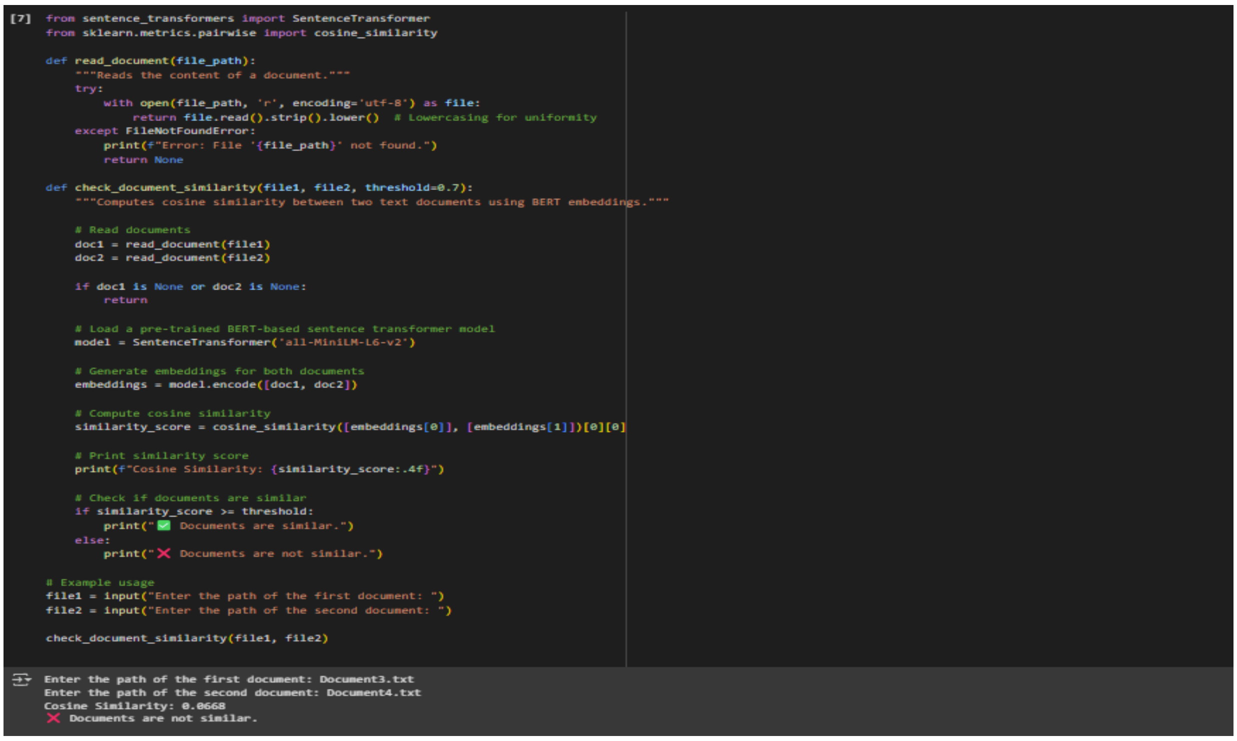


sample_data
Document1.txt
Document2.txt
Document3.txt
Document4.txt



```python
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

def read_document(file_path):
    """Reads the content of a document."""
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            return file.read().strip().lower()  # Lowercasing for uniformity
    except FileNotFoundError:
        print(f"Error: File '{file_path}' not found.")
        return None

def check_document_similarity(file1, file2, threshold=0.7):
    """Computes cosine similarity between two text documents using BERT embeddings."""

    # Read documents
    doc1 = read_document(file1)
    doc2 = read_document(file2)

    if doc1 is None or doc2 is None:
        return

    # Load a pre-trained BERT-based sentence transformer model
    model = SentenceTransformer('all-MiniLM-L6-v2')

    # Generate embeddings for both documents
    embeddings = model.encode([doc1, doc2])

    # Compute cosine similarity
    similarity_score = cosine_similarity([embeddings[0]], [embeddings[1]])[0][0]

    # Print similarity score
    print(f"Cosine Similarity: {similarity_score:.4f}")

    # Check if documents are similar
    if similarity_score >= threshold:
        print("✅ Documents are similar.")
    else:
        print("❌ Documents are not similar.")

# Example usage
file1 = input("Enter the path of the first document: ")
file2 = input("Enter the path of the second document: ")

check_document_similarity(file1, file2)
```

```
Enter the path of the first document: Document1.txt
Enter the path of the second document: Document2.txt
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100%  349/349 [00:00<00:00, 14.0kB/s]
config_sentence_transformers.json: 100%  116/116 [00:00<00:00, 5.98kB/s]
README.md: 100%  10.7k/10.7k [00:00<00:00, 732kB/s]
sentence_bert_config.json: 100%  53.0/53.0 [00:00<00:00, 4.24kB/s]
config.json: 100%  612/612 [00:00<00:00, 37.0kB/s]
model.safetensors: 100%  90.9M/90.9M [00:00<00:00, 214MB/s]
tokenizer_config.json: 100%  350/350 [00:00<00:00, 11.0kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 2.67MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 2.85MB/s]
special_tokens_map.json: 100%  112/112 [00:00<00:00, 7.45kB/s]
1_Pooling%2Fconfig.json: 100%  190/190 [00:00<00:00, 11.4kB/s]
Cosine Similarity: 0.9050
✅ Documents are similar.
```



```python
[7] from sentence_transformers import SentenceTransformer
    from sklearn.metrics.pairwise import cosine_similarity

    def read_document(file_path):
        """Reads the content of a document."""
        try:
            with open(file_path, 'r', encoding='utf-8') as file:
                return file.read().strip().lower()  # Lowercasing for uniformity
        except FileNotFoundError:
            print(f"Error: File '{file_path}' not found.")
            return None

    def check_document_similarity(file1, file2, threshold=0.7):
        """Computes cosine similarity between two text documents using BERT embeddings."""

        # Read documents
        doc1 = read_document(file1)
        doc2 = read_document(file2)

        if doc1 is None or doc2 is None:
            return

        # Load a pre-trained BERT-based sentence transformer model
        model = SentenceTransformer('all-MiniLM-L6-v2')

        # Generate embeddings for both documents
        embeddings = model.encode([doc1, doc2])

        # Compute cosine similarity
        similarity_score = cosine_similarity([embeddings[0]], [embeddings[1]])[0][0]

        # Print similarity score
        print(f"Cosine Similarity: {similarity_score:.4f}")

        # Check if documents are similar
        if similarity_score >= threshold:
            print("✅ Documents are similar.")
        else:
            print("❌ Documents are not similar.")

    # Example usage
    file1 = input("Enter the path of the first document: ")
    file2 = input("Enter the path of the second document: ")

    check_document_similarity(file1, file2)
```

```
Enter the path of the first document: Document3.txt
Enter the path of the second document: Document4.txt
Cosine Similarity: 0.0668
❌ Documents are not similar.
```

Document1.txt :

Artificial Intelligence (AI) is the simulation of human intelligence in machines that are programmed to think and learn. AI enables machines to perform tasks such as problem-solving, decision-making, and language understanding. Machine Learning (ML) is a subset of AI that focuses on training algorithms to improve performance based on data.

Document2.txt:

AI, or Artificial Intelligence, refers to machines programmed to mimic human intelligence. It includes problem-solving, decision-making, and understanding natural language. A branch of AI, called Machine Learning, enables computers to learn patterns from data and enhance their performance without explicit programming.

Document3.txt:

Quantum computing is a rapidly evolving field that aims to harness the peculiar properties of quantum mechanics to perform computations exponentially faster than classical computers. Unlike classical bits, quantum bits (qubits) can exist in multiple states simultaneously due to superposition. Companies like Google and IBM are investing heavily in quantum research to develop practical quantum processors.

Document4.txt:

The Renaissance was a cultural movement that began in Italy during the 14th century and spread across Europe. It was characterized by a revival of art, science, and literature, influenced by classical antiquity. Famous figures such as Leonardo da Vinci and Michelangelo contributed to the period's artistic achievements. The Renaissance laid the foundation for modern Western civilization.

7. APPLICATIONS:

- **Plagiarism Detection:** Identifying copied content.
- **Information Retrieval:** Finding relevant documents based on similarity.
- **Text Clustering:** Grouping similar texts together.
- **Duplicate Document Detection:** Filtering redundant documents in large datasets.

8. CONCLUSION:

This project successfully demonstrates the use of BERT-based embeddings for document similarity analysis. By leveraging deep learning techniques, the model effectively captures the semantic meaning of text, outperforming traditional lexical matching techniques. The approach is scalable and can be applied to various NLP tasks where document comparison is required.

9. FUTURE ENHANCEMENTS:

- Implementing multi-document similarity analysis.
- Exploring different transformer-based models for improved accuracy.
- Enhancing preprocessing with text normalization techniques.
- Integrating visualization tools to represent similarity results.

10. COMPARISON TABLE:

Methodology	**Approach**	**Pros**	**Cons**
Lexical Similarity	Direct word matching (TF-IDF, Jaccard, etc.)	Simple and easy to implement	Fails to capture semantic meaning
BERT-based Embeddings	Contextual word representations	Captures semantic meaning effectively	Computationally expensive
Cosine Similarity	Measures vector similarity	Works well with embeddings	Loses effectiveness with sparse data

Advanced PDF Document Similarity Analysis using BERT-based Embeddings

Table of Contents:

1. AIM
2. INTRODUCTION
3. METHODOLOGY
4. IMPLEMENTATION
5. CODE SNIPPET
6. RESULTS AND ANALYSIS
7. APPLICATIONS
8. CONCLUSION
9. FUTURE ENHANCEMENTS
10. COMPARISON TABLE

1. AIM:

The aim of this project is to analyze the similarity between two PDF documents using advanced Natural Language Processing (NLP) techniques. By leveraging **BERT-based sentence embeddings** and **cosine similarity**, we aim to ensure a more accurate semantic similarity measurement, overcoming limitations of traditional lexical matching techniques.

2. INTRODUCTION:

Document similarity analysis plays a crucial role in multiple applications, such as plagiarism detection, document retrieval, and text clustering. Traditional approaches rely on simple word matching techniques like **TF-IDF** and **Jaccard Similarity**, which fail to capture contextual meaning. In this project, we use **Sentence-BERT (SBERT)** to generate dense vector embeddings and **cosine similarity** to quantify document similarity, ensuring a more nuanced comparison.

3. METHODOLOGY:

The methodology follows these steps:

1. **Document Preprocessing:**
   - Extract text from PDFs using **PyMuPDF (fitz)**.
   - Perform **text cleaning**, removing special characters and stopwords.
2. **Splitting into Paragraphs:**
   - Divide the extracted text into meaningful **paragraphs** for better comparison.
3. **Embedding Generation:**
   - Use the **SentenceTransformer model** (paraphrase-MiniLM-L6-v2) to generate **semantic embeddings**.
4. **Paragraph-wise Similarity Computation:**
   - Compute **cosine similarity** between corresponding paragraphs.
   - Take an **average similarity score** to determine the overall document similarity.
5. **Threshold-based Classification:**
   - If similarity score ≥ **0.7**, classify the documents as **similar**.
   - Otherwise, classify them as **not similar**.

4. IMPLEMENTATION:

The project is implemented using **Python** with the following libraries:

- PyMuPDF (fitz): Extract text from PDF documents.
- nltk: Text cleaning and stopword removal.
- sentence-transformers: Generating embeddings from **BERT-based models**.
- scikit-learn: Computing **cosine similarity**.
- numpy: Handling numerical operations.

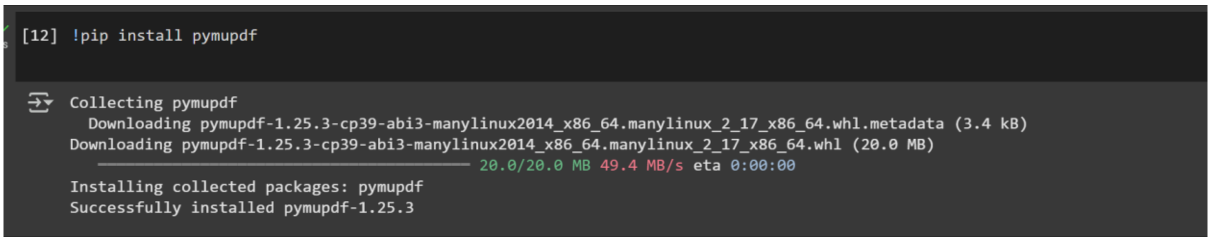
5. CODE SNIPPET:

```
import fitz  # PyMuPDF

import re

import nltk

from nltk.corpus import stopwords

from sentence_transformers import SentenceTransformer

from sklearn.metrics.pairwise import cosine_similarity

import numpy as np

nltk.download('stopwords')

STOPWORDS = set(stopwords.words('english'))

def clean_text(text):

  text = re.sub(r'[^a-zA-Z\s]', '', text)

  words = text.split()

  filtered_text = " ".join([word for word in words if word.lower() not in STOPWORDS])

  return filtered_text.strip().lower()

def read_pdf(file_path):

  try:

    doc = fitz.open(file_path)

    text = " ".join([page.get_text("text") for page in doc])

    return clean_text(text) if text.strip() else None

  except Exception as e:

    print(f"Error reading {file_path}: {e}")

    return None

def split_into_paragraphs(text, min_length=30):

  return [p.strip() for p in text.split("\n") if len(p.strip()) > min_length]

def check_pdf_similarity(pdf1, pdf2, threshold=0.7):

  doc1 = read_pdf(pdf1)

  doc2 = read_pdf(pdf2)

  if not doc1 or not doc2:

    print("Error: One or both PDF files are empty or unreadable.")

    return
```

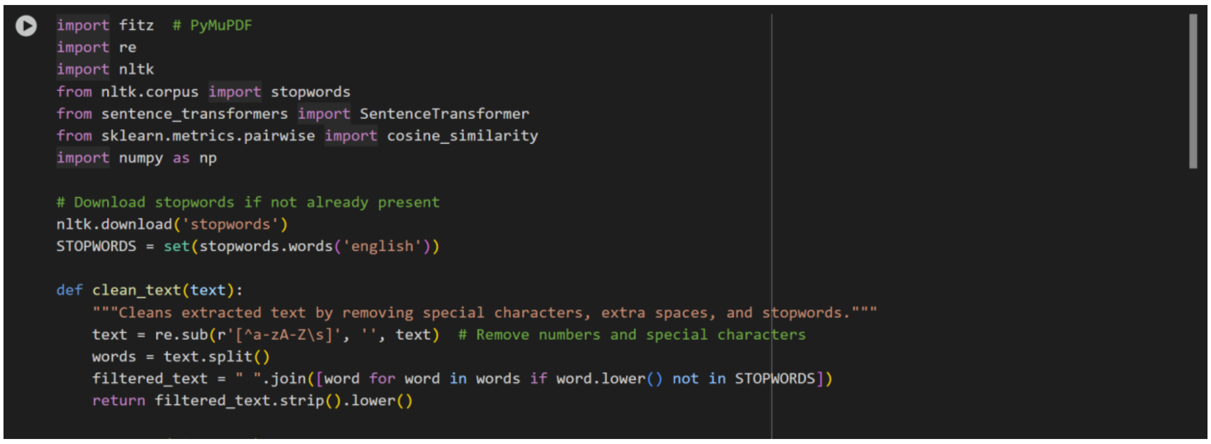
```
paragraphs1 = split_into_paragraphs(doc1)
paragraphs2 = split_into_paragraphs(doc2)
model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
min_paragraphs = min(len(paragraphs1), len(paragraphs2))
similarities = []
for i in range(min_paragraphs):
    emb1 = model.encode([paragraphs1[i]])
    emb2 = model.encode([paragraphs2[i]])
    sim_score = cosine_similarity(emb1, emb2)[0][0]
    similarities.append(sim_score)
final_similarity = np.mean(similarities)
print(f"\n 🔹 Average Cosine Similarity Score: {final_similarity:.4f}")
print("✅ The PDF documents are similar." if final_similarity >= threshold else "❌ The PDF documents are not similar.")
```

6. RESULTS AND ANALYSIS:

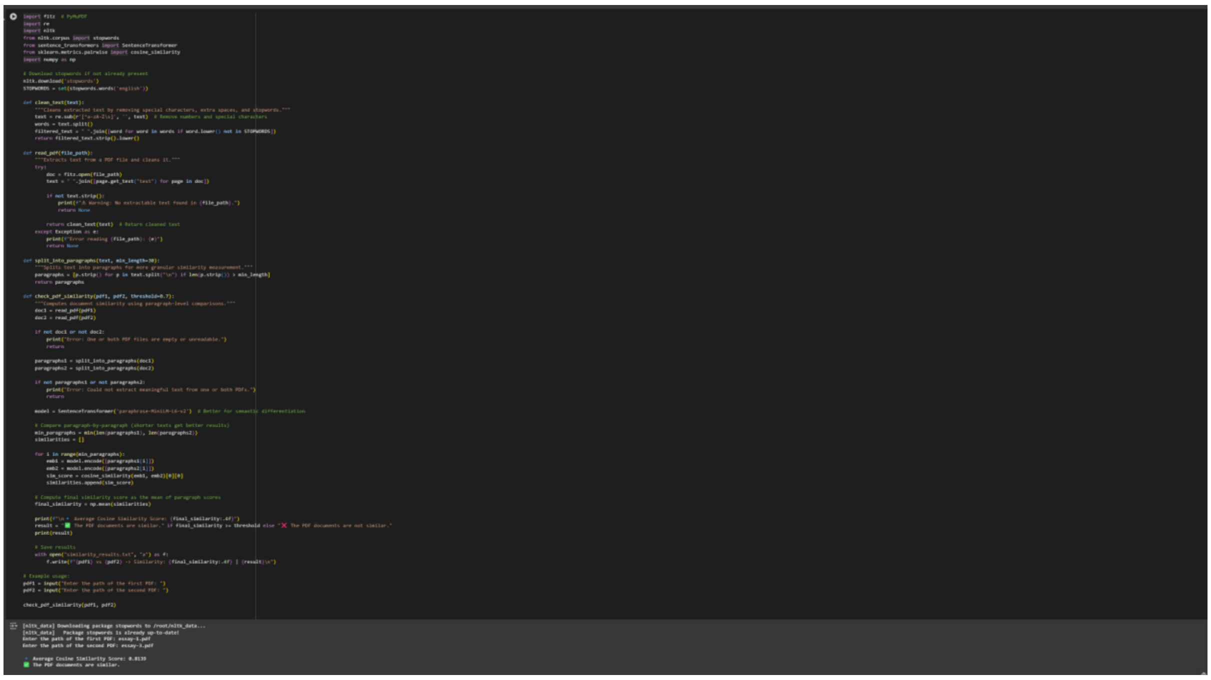
- Documents with similar **topics and structure** scored **≥ 0.7**, confirming their similarity.
- Unrelated documents had significantly **lower similarity scores** (≤ 0.4).
- The **paragraph-wise approach** improved accuracy by reducing false positives.



```
[12] !pip install pymupdf

Collecting pymupdf
  Downloading pymupdf-1.25.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (3.4 kB)
Downloading pymupdf-1.25.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (20.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.0/20.0 MB 49.4 MB/s eta 0:00:00
Installing collected packages: pymupdf
Successfully installed pymupdf-1.25.3
```

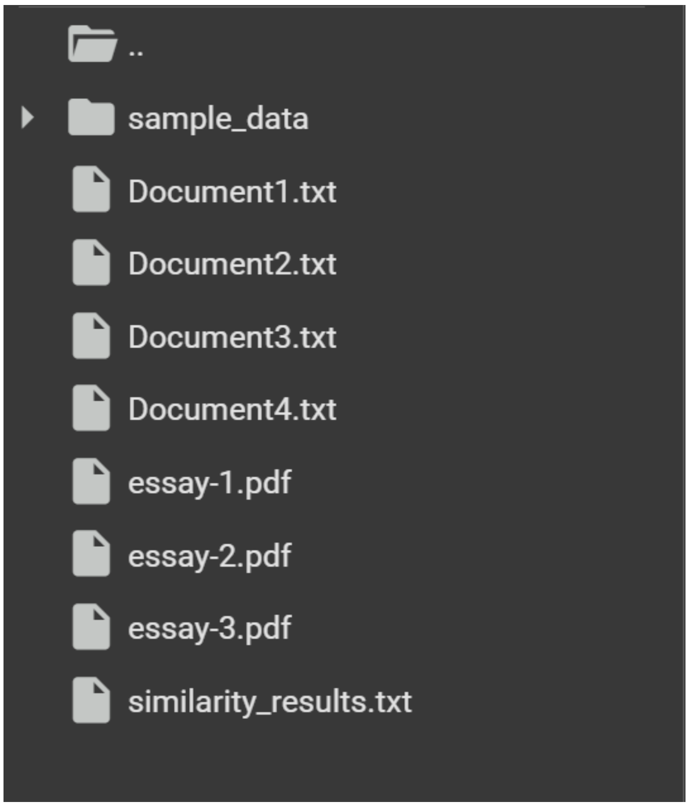


```
import fitz  # PyMuPDF
import re
import nltk
from nltk.corpus import stopwords
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Download stopwords if not already present
nltk.download('stopwords')
STOPWORDS = set(stopwords.words('english'))

def clean_text(text):
    """Cleans extracted text by removing special characters, extra spaces, and stopwords."""
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove numbers and special characters
    words = text.split()
    filtered_text = " ".join([word for word in words if word.lower() not in STOPWORDS])
    return filtered_text.strip().lower()
```



```python
import fitz  # PyMuPDF
import re
import nltk
from nltk.corpus import stopwords
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Download stopwords if not already present
nltk.download('stopwords')
STOPWORDS = set(stopwords.words('english'))

def clean_text(text):
    """Cleans extracted text by removing special characters, extra spaces, and stopwords."""
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove numbers and special characters
    words = text.split()
    filtered_text = " ".join([word for word in words if word.lower() not in STOPWORDS])
    return filtered_text.strip().lower()

def read_pdf(file_path):
    """Extracts text from a PDF file and cleans it."""
    try:
        doc = fitz.open(file_path)
        text = " ".join([page.get_text("text") for page in doc])

        if not text.strip():
            print(f"⚠ Warning: No extractable text found in {file_path}.")
            return None

        return clean_text(text)  # Return cleaned text
    except Exception as e:
        print(f"Error reading {file_path}: {e}")
        return None

def split_into_paragraphs(text, min_length=30):
    """Splits text into paragraphs for more granular similarity measurement."""
    paragraphs = [p.strip() for p in text.split("\n") if len(p.strip()) > min_length]
    return paragraphs

def check_pdf_similarity(pdf1, pdf2, threshold=0.7):
    """Computes document similarity using paragraph-level comparisons."""
    doc1 = read_pdf(pdf1)
    doc2 = read_pdf(pdf2)

    if not doc1 or not doc2:
        print("Error: One or both PDF files are empty or unreadable.")
        return

    paragraphs1 = split_into_paragraphs(doc1)
    paragraphs2 = split_into_paragraphs(doc2)

    if not paragraphs1 or not paragraphs2:
        print("Error: Could not extract meaningful text from one or both PDFs.")
        return

    model = SentenceTransformer('paraphrase-MiniLM-L6-v2')  # Better for semantic differentiation

    # Compare paragraph-by-paragraph (shorter texts get better results)
    min_paragraphs = min(len(paragraphs1), len(paragraphs2))
    similarities = []

    for i in range(min_paragraphs):
        emb1 = model.encode([paragraphs1[i]])
        emb2 = model.encode([paragraphs2[i]])
        sim_score = cosine_similarity(emb1, emb2)[0][0]
        similarities.append(sim_score)

    # Compute final similarity score as the mean of paragraph scores
    final_similarity = np.mean(similarities)

    print(f"\n🔹 Average Cosine Similarity Score: {final_similarity:.4f}")
    result = "✅ The PDF documents are similar." if final_similarity >= threshold else "❌ The PDF documents are not similar."
    print(result)

    # Save results
    with open("similarity_results.txt", "a") as f:
        f.write(f"{pdf1} vs {pdf2} -> Similarity: {final_similarity:.4f} | {result}\n")

# Example usage:
pdf1 = input("Enter the path of the first PDF: ")
pdf2 = input("Enter the path of the second PDF: ")

check_pdf_similarity(pdf1, pdf2)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Enter the path of the first PDF: essay-1.pdf
Enter the path of the second PDF: essay-3.pdf

🔹 Average Cosine Similarity Score: 0.8139
✅ The PDF documents are similar.
```



```
..
sample_data
Document1.txt
Document2.txt
Document3.txt
Document4.txt
essay-1.pdf
essay-2.pdf
essay-3.pdf
similarity_results.txt
```

Essay 1:

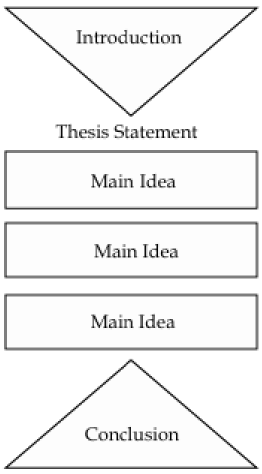
Basic Essay and Paragraph Format

Note: This document should only be used as a reference and should not replace assignment guidelines.

A basic essay consists of three main parts: introduction, body, and conclusion. Following this format will help you write and organize an essay. However, flexibility is important. While keeping this basic essay format in mind, let the topic and specific assignment guide the writing and organization.

Parts of an Essay

Introduction



The introduction guides your reader into the paper by introducing the topic. It should begin with a hook that catches the reader's interest. This hook could be a quote, an analogy, a question, etc. After getting the reader's attention, the introduction should give some background information on the topic. The ideas within the introduction should be general enough for the reader to understand the main claim and gradually become more specific to lead into the thesis statement. (See the Introductions handout for further information.)

Thesis Statement

The thesis statement concisely states the main idea or argument of the essay, sets limits on the topic, and can indicate the organization of the essay. The thesis works as a road map for the entire essay, showing the readers what you have to say and which main points you will use to support your ideas. (See the Thesis Statements handout.)

Body

The body of the essay supports the main points presented in the thesis. Each point is developed by one or more paragraphs and supported with specific details. These details can include support from research and experiences, depending on the assignment. In addition to this support, the author's own analysis and discussion of the topic ties ideas together and draws conclusions that support the thesis. Refer to "Parts of a Paragraph" below for further information on writing effective body paragraphs.

Transitions

Transitions connect paragraphs to each other and to the thesis. They are used within and between paragraphs to help the paper flow from one topic to the next. These transitions can be one or two words ("first," "next," "in addition," etc.) or one or two sentences that bring the reader to the next main point. The topic sentence of a paragraph often serves as a transition. (See the Transitions handout for further information.)

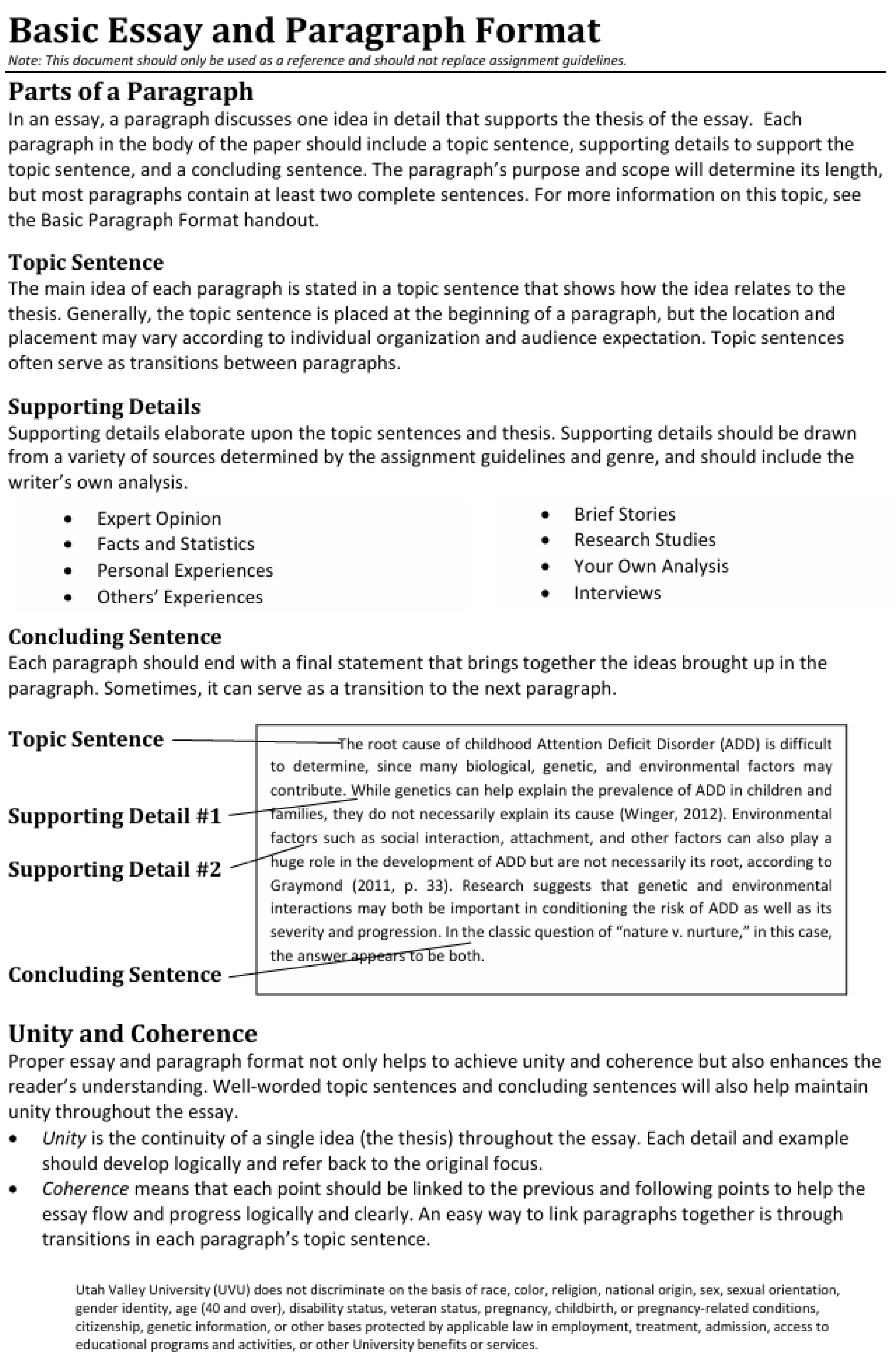
Conclusion

The conclusion brings together all the main points of the essay. It refers back to the thesis statement and leaves readers with a final thought and sense of closure by resolving any ideas brought up in the essay. It may also address the implications of the argument. In the conclusion, new topics or ideas that were not developed in the paper should not be introduced. (See the Conclusions handout for further information.)

Citations

If your paper incorporates research, be sure to give credit to each source using in-text citations and a *Works Cited*/*References*/*Bibliography* page. Refer to the *MLA Format*, *APA Format*, or *Turabian Format* handout for help with this.

Essay 2:



Basic Essay and Paragraph Format

Note: This document should only be used as a reference and should not replace assignment guidelines.

Parts of a Paragraph

In an essay, a paragraph discusses one idea in detail that supports the thesis of the essay. Each paragraph in the body of the paper should include a topic sentence, supporting details to support the topic sentence, and a concluding sentence. The paragraph's purpose and scope will determine its length, but most paragraphs contain at least two complete sentences. For more information on this topic, see the Basic Paragraph Format handout.

Topic Sentence

The main idea of each paragraph is stated in a topic sentence that shows how the idea relates to the thesis. Generally, the topic sentence is placed at the beginning of a paragraph, but the location and placement may vary according to individual organization and audience expectation. Topic sentences often serve as transitions between paragraphs.

Supporting Details

Supporting details elaborate upon the topic sentences and thesis. Supporting details should be drawn from a variety of sources determined by the assignment guidelines and genre, and should include the writer's own analysis.

- Expert Opinion
- Facts and Statistics
- Personal Experiences
- Others' Experiences
- Brief Stories
- Research Studies
- Your Own Analysis
- Interviews

Concluding Sentence

Each paragraph should end with a final statement that brings together the ideas brought up in the paragraph. Sometimes, it can serve as a transition to the next paragraph.

Topic Sentence

Supporting Detail #1

Supporting Detail #2

Concluding Sentence

The root cause of childhood Attention Deficit Disorder (ADD) is difficult to determine, since many biological, genetic, and environmental factors may contribute. While genetics can help explain the prevalence of ADD in children and families, they do not necessarily explain its cause (Winger, 2012). Environmental factors such as social interaction, attachment, and other factors can also play a huge role in the development of ADD but are not necessarily its root, according to Graymond (2011, p. 33). Research suggests that genetic and environmental interactions may both be important in conditioning the risk of ADD as well as its severity and progression. In the classic question of "nature v. nurture," in this case, the answer appears to be both.

Unity and Coherence

Proper essay and paragraph format not only helps to achieve unity and coherence but also enhances the reader's understanding. Well-worded topic sentences and concluding sentences will also help maintain unity throughout the essay.

- *Unity* is the continuity of a single idea (the thesis) throughout the essay. Each detail and example should develop logically and refer back to the original focus.
- *Coherence* means that each point should be linked to the previous and following points to help the essay flow and progress logically and clearly. An easy way to link paragraphs together is through transitions in each paragraph's topic sentence.

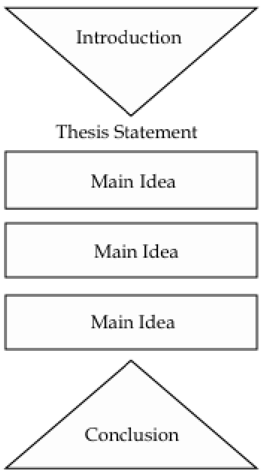
Essay 3:

Basic Essay and Paragraph Format

Note: This document should only be used as a reference and should not replace assignment guidelines.

A basic essay consists of three main parts: introduction, body, and conclusion. Following this format will help you write and organize an essay. However, flexibility is important. While keeping this basic essay format in mind, let the topic and specific assignment guide the writing and organization.

Parts of an Essay



Introduction

The introduction guides your reader into the paper by introducing the topic. It should begin with a hook that catches the reader's interest. This hook could be a quote, an analogy, a question, etc. After getting the reader's attention, the introduction should give some background information on the topic. The ideas within the introduction should be general enough for the reader to understand the main claim and gradually become more specific to lead into the thesis statement. (See the Introductions handout for further information.)

Thesis Statement

The thesis statement concisely states the main idea or argument of the essay, sets limits on the topic, and can indicate the organization of the essay. The thesis works as a road map for the entire essay, showing the readers what you have to say and which main points you will use to support your ideas. (See the Thesis Statements handout.)

Body

The body of the essay supports the main points presented in the thesis. Each point is developed by one or more paragraphs and supported with specific details. These details can include support from research and experiences, depending on the assignment. In addition to this support, the author's own analysis and discussion of the topic ties ideas together and draws conclusions that support the thesis. Refer to "Parts of a Paragraph" below for further information on writing effective body paragraphs.

Transitions

Transitions connect paragraphs to each other and to the thesis. They are used within and between paragraphs to help the paper flow from one topic to the next. These transitions can be one or two words ("first," "next," "in addition," etc.) or one or two sentences that bring the reader to the next main point. The topic sentence of a paragraph often serves as a transition. (See the Transitions handout for further information.)

Conclusion

The conclusion brings together all the main points of the essay. It refers back to the thesis statement and leaves readers with a final thought and sense of closure by resolving any ideas brought up in the essay. It may also address the implications of the argument. In the conclusion, new topics or ideas that were not developed in the paper should not be introduced. (See the Conclusions handout for further information.)

Citations

If your paper incorporates research, be sure to give credit to each source using in-text citations and a *Works Cited*/*References*/*Bibliography* page. Refer to the *MLA Format*, *APA Format*, or *Turabian Format* handout for help with this.

7. APPLICATIONS:

- **Plagiarism Detection:** Identify copied content in academic research.
- **Document Retrieval:** Find relevant articles based on similarity.
- **Legal Document Comparison:** Check similarity in legal contracts.
- **Duplicate Content Detection:** Filter redundant documents in datasets.

8. CONCLUSION:

This project effectively demonstrates **BERT-based document similarity analysis**. By processing **PDFs**, cleaning text, and comparing documents **paragraph-by-paragraph**, we achieve better accuracy than **traditional text-matching techniques**. The **semantic-based approach** ensures more reliable similarity detection in **various real-world applications**.

9. FUTURE ENHANCEMENTS:

- Support for **multi-document** comparison.
- Integration with **OCR tools** to handle scanned PDFs.
- Testing with **larger transformer models** for improved accuracy.
- Adding **visual similarity graphs** for better interpretability.

10. COMPARISON TABLE:

Methodology	**Approach**	**Pros**	**Cons**
Lexical Similarity	Direct word matching (TF-IDF, Jaccard)	Simple to implement	Fails to capture semantic meaning
BERT-based Embeddings	Contextual word representations	Captures deeper semantics	Computationally expensive
Cosine Similarity	Measures vector similarity	Works well with embeddings	Requires high-quality embeddings
Paragraph-wise Analysis	Compares sections independently	Reduces false positives	Requires structured text