**CHAPTER 3:**

**fixed\_size\_chunking.py**

*def fixed\_size\_chunking(text, chunk\_size=45):*

*return [text[i:i+chunk\_size] for i in range(0, len(text), chunk\_size)]*

*chunks\_fixed = fixed\_size\_chunking(text)*

*# To print the first chunk*

*print(chunks\_fixed[0])*

**sentence\_splitting.py**

*import nltk*

*nltk.download('punkt')*

*from nltk.tokenize import sent\_tokenize*

*def sentence\_splitting(text, max\_tokens=30):*

*sentences = sent\_tokenize(text)*

*chunks = []*

*current\_chunk = []*

*current\_length = 0*

*for sentence in sentences:*

*if current\_length + len(sentence.split()) <= max\_tokens:*

*current\_chunk.append(sentence)*

*current\_length += len(sentence.split())*

*else:*

*chunks.append(' '.join(current\_chunk))*

*current\_chunk = [sentence]*

*current\_length = len(sentence.split())*

*if current\_chunk:*

*chunks.append(' '.join(current\_chunk))*

*return chunks*

*chunks\_sentence = sentence\_splitting(text)*

*# To print the first chunk*

*print(chunks\_sentence[0])*

#To view the entire chunking

chunks\_sentence

**recursive\_chunking.py**

*def recursive\_chunking(text, max\_tokens=30):*

*if len(text.split()) <= max\_tokens:*

*return [text]*

*else:*

*midpoint = len(text) // 2*

*first\_half = text[:midpoint]*

*second\_half = text[midpoint:]*

*return recursive\_chunking(first\_half, max\_tokens) + recursive\_chunking(second\_half, max\_tokens)*

*chunks\_recursive = recursive\_chunking(text)*

*# To print the first chunk*

*print(chunks\_recursive[0])*

*#To view the entire chunking*

*chunks\_recursive*

**embeddings.py**

*from transformers import AutoTokenizer, AutoModel*

*# Select pre-trained model*

*model\_name = "bert-base-uncased"*

*tokenizer = AutoTokenizer.from\_pretrained(model\_name)*

*model = AutoModel.from\_pretrained(model\_name)*

*# Function to create the text embeddings*

*def get\_embeddings(text):*

*model\_name = "bert-base-uncased"*

*tokenizer = AutoTokenizer.from\_pretrained(model\_name)*

*model = AutoModel.from\_pretrained(model\_name)*

*tokens = tokenizer(text, return\_tensors="pt")*

*outputs = model(\*\*tokens)*

*embeddings = outputs.last\_hidden\_state*

*print(embeddings)*

*tensor = embeddings.detach().numpy()*

*#Reshaping text1*

*nsamples, nx, ny = tensor.shape*

*tensor = tensor.reshape((nsamples,nx\*ny))*

*one\_D\_embed = np.reshape(tensor, (np.product(tensor.shape),))*

*print(one\_D\_embed)*

*return one\_D\_embed*

**similiarity\_score.py**

*text1 ="The Transformer can be trained significantly faster for translation tasks."*

*text2 = "The Transformer architecture helps it to excel at translation tasks."*

*import numpy as np*

*from sklearn.metrics.pairwise import cosine\_similarity*

*from transformers import AutoTokenizer, AutoModel*

*# Tokenization*

*tokens1 = tokenizer(text1, return\_tensors="pt")*

*tokens2 = tokenizer(text2, return\_tensors="pt")*

*# Retrieve the embeddings*

*outputs1 = model(\*\*tokens1)*

*embeddings1 = outputs1.last\_hidden\_state*

*outputs2 = model(\*\*tokens2)*

*embeddings2 = outputs2.last\_hidden\_state*

*#Reshaping the array to compute the similarity score*

*tensor1 = embeddings1.detach().numpy()*

*tensor2 = embeddings2.detach().numpy()*

*#Reshaping text1*

*nsamples, nx, ny = tensor1.shape*

*tensor11 = tensor1.reshape((nsamples,nx\*ny))*

*#Reshaping text2*

*nsamples, nx, ny = tensor2.shape*

*tensor22 = tensor2.reshape((nsamples,nx\*ny))*

*# Calculate cosine similarity between the embeddings*

*similarity\_score = cosine\_similarity(tensor11, tensor22)*

*print("Similarity Score between two sentences:", similarity\_score[0][0])*