

1)

Sample:

Patty was far from being conceited over her pretty face, but she honestly liked admiration, and, indeed, she was accustomed to receive it from all who knew her. At the present moment, she was standing before a long mirror in her boudoir, putting the last touches to her new party toilette. Louise, the maid, stood by, with a fur-trimmed wrap, and Patty drew on her long gloves with a happy smile of anticipation.

Manually (without PSO):

Temperature: 1.0, Length: 100, METEOR Score: 0.9747

Generated Text:

Louise, the maid, stood by, with a fur-trimmed wrap, and Patty drew on her long gloves with a happy smile of anticipation. Patty was far from being conceited over her pretty face, but she honestly liked admiration, and, indeed, she was accustomed to receive it from all who knew her. At the present moment, she was standing before a long mirror in her boudoir, putting the last touches to her new party toilette.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')  # Only 'punkt' for sentence tokenization

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')  # Ensure NLTK looks in the correct directory

# New original passage

original\_text = """Patty was far from being conceited over her pretty face, but she honestly liked admiration,

and, indeed, she was accustomed to receive it from all who knew her. At the present moment, she was standing before

a long mirror in her boudoir, putting the last touches to her new party toilette. Louise, the maid, stood by, with

a fur-trimmed wrap, and Patty drew on her long gloves with a happy smile of anticipation."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature (higher = more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Stopping search: maximum iterations reached --> 100

Optimized Temperature: 0.8031, METEOR Score: 0.9844

Generated Text:

Patty was far from being conceited over her pretty face, but she honestly liked admiration, and, indeed, she was accustomed to receive it from all who knew her. Louise, the maid, stood by, with a fur-trimmed wrap, and Patty drew on her long gloves with a happy smile of anticipation. At the present moment, she was standing before a long mirror in her boudoir, putting the last touches to her new party toilette.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

import numpy as np

from pyswarm import pso  # PSO optimization

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Patty was far from being conceited over her pretty face, but she honestly liked admiration,

and, indeed, she was accustomed to receive it from all who knew her. At the present moment, she was standing before

a long mirror in her boudoir, putting the last touches to her new party toilette. Louise, the maid, stood by, with

a fur-trimmed wrap, and Patty drew on her long gloves with a happy smile of anticipation."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with a given temperature

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO (minimizing METEOR score)

def objective\_function(temp):

    generated\_text = generate\_response(original\_text, temperature=temp[0], response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score  # Minimize METEOR score

# PSO Optimization

lb = [0.01]  # Lower bound for temperature

ub = [1.0]   # Upper bound for temperature

best\_temp, \_ = pso(objective\_function, lb, ub)

# Generate text using optimized temperature

generated\_text = generate\_response(original\_text, temperature=best\_temp[0], response\_length=100)

generated\_tokens = generated\_text.split()

# Compute METEOR score for best temperature

optimized\_score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Optimized Temperature: {best\_temp[0]:.4f}, METEOR Score: {optimized\_score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)

2)

Sample:

The telephone receiver at Dotty's end of the wire was hung up with a click, and Dolly began to waggle her receiver hook in hope of getting Dotty back. But there was no response, so Dolly rose and went for her coat. Flinging it round her, and not stopping to get a hat, she ran next door to Dotty Rose's house.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9796

Generated Text:

Flinging it round her, and not stopping to get a hat, she ran next door to Dotty Rose's house. The telephone receiver at Dotty's end of the wire was hung up with a click, and Dolly began to waggle her receiver hook in hope of getting Dotty back. But there was no response, so Dolly rose and went for her coat.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')  # Only 'punkt' for sentence tokenization

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')  # Ensure NLTK looks in the correct directory

# New original passage

original\_text = """The telephone receiver at Dotty's end of the wire was hung up with a click,

and Dolly began to waggle her receiver hook in hope of getting Dotty back.

But there was no response, so Dolly rose and went for her coat.

Flinging it round her, and not stopping to get a hat, she ran next door to Dotty Rose's house."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature (higher = more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Stopping search: maximum iterations reached --> 100

Optimized Temperature: 0.2208, METEOR Score: 0.9344

Generated Text:

Flinging it round her, and not stopping to get a hat, she ran next door to Dotty Rose's house. But there was no response, so Dolly rose and went for her coat. The telephone receiver at Dotty's end of the wire was hung up with a click, and Dolly began to waggle her receiver hook in hope of getting Dotty back.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

import numpy as np

from pyswarm import pso  # PSO optimization library

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')  # Only 'punkt' for sentence tokenization

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')  # Ensure NLTK looks in the correct directory

# New original passage

original\_text = """The telephone receiver at Dotty's end of the wire was hung up with a click,

and Dolly began to waggle her receiver hook in hope of getting Dotty back.

But there was no response, so Dolly rose and went for her coat.

Flinging it round her, and not stopping to get a hat, she ran next door to Dotty Rose's house."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature (higher = more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO (minimizing METEOR score)

def objective\_function(temp):

    generated\_text = generate\_response(original\_text, temperature=temp[0], response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score  # Minimize METEOR score

# PSO Optimization

lb = [0.01]  # Lower bound for temperature

ub = [1.0]   # Upper bound for temperature

best\_temp, \_ = pso(objective\_function, lb, ub)

# Generate text using optimized temperature

generated\_text = generate\_response(original\_text, temperature=best\_temp[0], response\_length=100)

generated\_tokens = generated\_text.split()

# Compute METEOR score for best temperature

optimized\_score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Optimized Temperature: {best\_temp[0]:.4f}, METEOR Score: {optimized\_score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # Print the generated text

3)

Sample:

The girls were down at Lakewood. Patty was the guest of Elise, whose family had taken a cottage there for the season. That is, it was called a cottage, but was in reality an immense house, most comfortably and delightfully appointed. Patty was still supposed to be convalescing from her recent illness, but, as a matter of fact, she had regained her health and strength, and, though never robust, was entirely well.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9561

Generated Text:

Patty was still supposed to be convalescing from her recent illness, but, as a matter of fact, she had regained her health and strength, and, though never robust, was entirely well. That is, it was called a cottage, but was in reality an immense house, most comfortably and delightfully appointed. Patty was the guest of Elise, whose family had taken a cottage there for the season. The girls were down at Lakewood.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('punkt')

nltk.download('wordnet')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The girls were down at Lakewood. Patty was the guest of Elise, whose family had taken a cottage there for the season.

That is, it was called a cottage, but was in reality an immense house, most comfortably and delightfully appointed.

Patty was still supposed to be convalescing from her recent illness, but, as a matter of fact, she had regained her health and strength,

and, though never robust, was entirely well."""

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature (higher = more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.4877, METEOR Score: 0.9922

Generated Text:

That is, it was called a cottage, but was in reality an immense house, most comfortably and delightfully appointed. Patty was the guest of Elise, whose family had taken a cottage there for the season. The girls were down at Lakewood. Patty was still supposed to be convalescing from her recent illness, but, as a matter of fact, she had regained her health and strength, and, though never robust, was entirely well.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

import numpy as np

from pyswarm import pso  # PSO optimization library

# Ensure necessary resources are downloaded

nltk.download('punkt')

nltk.download('wordnet')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The girls were down at Lakewood. Patty was the guest of Elise, whose family had taken a cottage there for the season.

That is, it was called a cottage, but was in reality an immense house, most comfortably and delightfully appointed.

Patty was still supposed to be convalescing from her recent illness, but, as a matter of fact, she had regained her health and strength,

and, though never robust, was entirely well."""

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature (higher = more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO (minimizing METEOR score)

def objective\_function(temp):

    generated\_text = generate\_response(original\_text, temperature=temp[0], response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score  # Minimize METEOR score

# PSO Optimization

lb = [0.01]  # Lower bound for temperature

ub = [1.0]   # Upper bound for temperature

best\_temp, \_ = pso(objective\_function, lb, ub)

# Generate text using optimized temperature

generated\_text = generate\_response(original\_text, temperature=best\_temp[0], response\_length=100)

generated\_tokens = generated\_text.split()

# Compute METEOR score for best temperature

optimized\_score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Optimized Temperature: {best\_temp[0]:.4f}, METEOR Score: {optimized\_score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # Print the generated text

4)

Sample:

One day a smart little groom rode into the court where Tom lived. Tom was just hiding behind a wall, to heave half a brick at his horse's legs, as is the custom of that country when they welcome strangers; but the groom saw him, and halloed to him to know where Mr. Grimes, the chimney-sweep, lived. Now, Mr. Grimes was Tom's own master, and Tom was a good man of business, and always civil to customers, so he put the half-brick down quietly behind the wall, and proceeded to take orders.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9887

Generated Text:

Tom was just hiding behind a wall, to heave half a brick at his horse's legs, as is the custom of that country when they welcome strangers; but the groom saw him, and halloed to him to know where Mr. Grimes, the chimney-sweep, lived. One day a smart little groom rode into the court where Tom lived. Now, Mr. Grimes was Tom's own master, and Tom was a good man of business, and always civil to customers, so he put the half-brick down quietly behind the wall, and proceeded to take orders.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')  # Only 'punkt' for sentence tokenization

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')  # Ensure NLTK looks in the correct directory

# New original passage

original\_text = """One day a smart little groom rode into the court where Tom lived.

Tom was just hiding behind a wall, to heave half a brick at his horse's legs, as is the custom of that country when they welcome strangers;

but the groom saw him, and halloed to him to know where Mr. Grimes, the chimney-sweep, lived.

Now, Mr. Grimes was Tom's own master, and Tom was a good man of business, and always civil to customers, so he put the half-brick down quietly behind the wall, and proceeded to take orders."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature (higher = more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.8267, METEOR Score: 0.9887

Generated Text:

Tom was just hiding behind a wall, to heave half a brick at his horse's legs, as is the custom of that country when they welcome strangers; but the groom saw him, and halloed to him to know where Mr. Grimes, the chimney-sweep, lived. One day a smart little groom rode into the court where Tom lived. Now, Mr. Grimes was Tom's own master, and Tom was a good man of business, and always civil to customers, so he put the half-brick down quietly behind the wall, and proceeded to take orders.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure it's installed with: pip install pyswarm

# Download necessary NLTK resources

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """One day a smart little groom rode into the court where Tom lived.

Tom was just hiding behind a wall, to heave half a brick at his horse's legs, as is the custom of that country when they welcome strangers;

but the groom saw him, and halloed to him to know where Mr. Grimes, the chimney-sweep, lived.

Now, Mr. Grimes was Tom's own master, and Tom was a good man of business, and always civil to customers, so he put the half-brick down quietly behind the wall, and proceeded to take orders."""

# Function to generate response based on temperature and length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature to control randomness

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function to be minimized (negative METEOR score)

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return -score  # Minimize the negative score

# Correct bounds: temperature in (0, 1]

lb = [0.01]  # Prevents zero temperature

ub = [1.0]   # Ensures temp does not exceed 1.0

# Run Particle Swarm Optimization

best\_temp, \_ = pso(objective\_function, lb, ub)

optimized\_temp = best\_temp[0]

# Generate final output using optimized temperature

generated\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

generated\_tokens = generated\_text.split()

final\_score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\n Generated Text:\n")

print(generated\_text)

5)

Sample:

Now it came to pass that in time Danae bore a son; so beautiful a babe that any but King Acrisius would have had pity on it.  But he had no pity; for he took Danae and her babe down to the seashore, and put them into a great chest and thrust them out to sea, for the winds and the waves to carry them whithersoever they would.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9571

Generated Text:

But he had no pity; for he took Danae and her babe down to the seashore, and put them into a great chest and thrust them out to sea, for the winds and the waves to carry them whithersoever they would. Now it came to pass that in time Danae bore a son; so beautiful a babe that any but King Acrisius would have had pity on it.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')  # Only 'punkt' for sentence tokenization

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')  # Ensure NLTK looks in the correct directory

# New original passage

original\_text = """Now it came to pass that in time Danae bore a son; so beautiful a babe that any but King Acrisius would have had pity on it.

But he had no pity; for he took Danae and her babe down to the seashore, and put them into a great chest and thrust them out to sea,

for the winds and the waves to carry them whithersoever they would."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature (higher = more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5614, METEOR Score: 1.0000

Generated Text:

Now it came to pass that in time Danae bore a son; so beautiful a babe that any but King Acrisius would have had pity on it. But he had no pity; for he took Danae and her babe down to the seashore, and put them into a great chest and thrust them out to sea, for the winds and the waves to carry them whithersoever they would.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure it's installed with: pip install pyswarm

# Download required NLTK data

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Now it came to pass that in time Danae bore a son; so beautiful a babe that any but King Acrisius would have had pity on it.

But he had no pity; for he took Danae and her babe down to the seashore, and put them into a great chest and thrust them out to sea,

for the winds and the waves to carry them whithersoever they would."""

# Function to generate response based on temperature and length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature to randomize sentence order

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function to be minimized (negative METEOR)

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return -score  # Minimize the negative score

# Correct bounds: temperature in (0.01, 1.0]

lb = [0.01]

ub = [1.0]

# Run Particle Swarm Optimization

best\_temp, \_ = pso(objective\_function, lb, ub)

optimized\_temp = best\_temp[0]

# Generate final output using optimized temperature

generated\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

generated\_tokens = generated\_text.split()

final\_score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\n Generated Text:\n")

print(generated\_text)

6)

Sample:

Once upon a time there were Three Bears who lived together in a house of their own in a wood. One of them was a Little, Small, Wee Bear; and one was a Middle-sized Bear, and the other was a Great, Huge Bear. They had each a pot for their porridge; a little pot for the Little, Small, Wee Bear; and a middle-sized pot for the Middle Bear; and a great pot for the Great, Huge Bear. And they had each a chair to sit in; a little chair for the Little, Small, Wee Bear; and a middle-sized chair for the Middle Bear; and a great chair for the Great, Huge Bear. And they had each a bed to sleep in; a little bed for the Little, Small, Wee Bear; and a middle-sized bed for the Middle Bear; and a great bed for the Great, Huge Bear.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9412

Generated Text:

Once upon a time there were Three Bears who lived together in a house of their own in a wood. And they had each a bed to sleep in; a little bed for the Little, Small, Wee Bear; and a middle-sized bed for the Middle Bear; and a great bed for the Great, Huge Bear. And they had each a chair to sit in; a little chair for the Little, Small, Wee Bear; and a middle-sized chair for the Middle Bear; and a great chair for the Great, Huge Bear. One of them was a Little, Small, Wee Bear; and one was a Middle-sized Bear, and the other was a Great, Huge Bear. They had each a pot for their porridge; a little pot for the Little, Small, Wee Bear; and a middle-sized pot for the Middle Bear; and a great pot for the Great, Huge Bear.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')  # Only 'punkt' for sentence tokenization

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')  # Ensure NLTK looks in the correct directory

# New original passage

original\_text = """Once upon a time there were Three Bears who lived together in a house of their own in a wood.

One of them was a Little, Small, Wee Bear; and one was a Middle-sized Bear, and the other was a Great, Huge Bear.

They had each a pot for their porridge; a little pot for the Little, Small, Wee Bear; and a middle-sized pot for the Middle Bear;

and a great pot for the Great, Huge Bear. And they had each a chair to sit in; a little chair for the Little, Small, Wee Bear;

and a middle-sized chair for the Middle Bear; and a great chair for the Great, Huge Bear. And they had each a bed to sleep in;

a little bed for the Little, Small, Wee Bear; and a middle-sized bed for the Middle Bear; and a great bed for the Great, Huge Bear."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature (higher = more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.8422

METEOR Score: 0.9253

Generated Text:

And they had each a chair to sit in; a little chair for the Little, Small, Wee Bear; and a middle-sized chair for the Middle Bear; and a great chair for the Great, Huge Bear. One of them was a Little, Small, Wee Bear; and one was a Middle-sized Bear, and the other was a Great, Huge Bear. And they had each a bed to sleep in; a little bed for the Little, Small, Wee Bear; and a middle-sized bed for the Middle Bear; and a great bed for the Great, Huge Bear. Once upon a time there were Three Bears who lived together in a house of their own in a wood. They had each a pot for their porridge; a little pot for the Little, Small, Wee Bear; and a middle-sized pot for the Middle Bear; and a great pot for the Great, Huge Bear.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # install via `pip install pyswarm` if needed

# Ensure necessary NLTK resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

nltk.data.path.append('./nltk\_data')

# Original text sample

original\_text = """Once upon a time there were Three Bears who lived together in a house of their own in a wood.

One of them was a Little, Small, Wee Bear; and one was a Middle-sized Bear, and the other was a Great, Huge Bear.

They had each a pot for their porridge; a little pot for the Little, Small, Wee Bear; and a middle-sized pot for the Middle Bear;

and a great pot for the Great, Huge Bear. And they had each a chair to sit in; a little chair for the Little, Small, Wee Bear;

and a middle-sized chair for the Middle Bear; and a great chair for the Great, Huge Bear. And they had each a bed to sleep in;

a little bed for the Little, Small, Wee Bear; and a middle-sized bed for the Middle Bear; and a great bed for the Great, Huge Bear."""

# Function to generate sentence-level response

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function to minimize (-METEOR score)

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return -score  # minimize the negative METEOR score

# Corrected bounds: temperature strictly between 0.0 and 1.0

lb = [0.01]  # avoid 0 to prevent division errors

ub = [1.0]

# Run PSO

best\_temp, \_ = pso(objective\_function, lb, ub)

optimized\_temp = best\_temp[0]

# Generate and evaluate final result

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Output

print(f"Optimized Temperature: {optimized\_temp:.4f}")

print(f"METEOR Score: {final\_score:.4f}")

print("\n Generated Text:\n")

print(final\_text)

7)

Sample:

The little body of British troopers, only forty-eight of them all told, with Hal Paine and Chester Crawford as their guides, were reconnoitering ten miles in advance of the main army along the river Marne in the great war between Germany and the allied armies. For several hours they had been riding slowly without encountering the enemy, when, suddenly, as the little squad topped a small hill and the two boys gained an unobstructed view of the little plain below, Hal pulled up his horse with an exclamation.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9456

Generated Text:

For several hours they had been riding slowly without encountering the enemy, when, suddenly, as the little squad topped a small hill and the two boys gained an unobstructed view of the little plain below, Hal pulled up his horse with an exclamation. The little body of British troopers, only forty-eight of them all told, with Hal Paine and Chester Crawford as their guides, were reconnoitering ten miles in advance of the main army along the river Marne in the great war between Germany and the allied armies.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The little body of British troopers, only forty-eight of them all told, with Hal Paine

and Chester Crawford as their guides, were reconnoitering ten miles in advance of the main army along the river

Marne in the great war between Germany and the allied armies. For several hours they had been riding slowly

without encountering the enemy, when, suddenly, as the little squad topped a small hill and the two boys gained

an unobstructed view of the little plain below, Hal pulled up his horse with an exclamation."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment only when it's less than 1.0

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature is between 0 and 1

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.4222, METEOR Score: 0.9456

Generated Text:

For several hours they had been riding slowly without encountering the enemy, when, suddenly, as the little squad topped a small hill and the two boys gained an unobstructed view of the little plain below, Hal pulled up his horse with an exclamation. The little body of British troopers, only forty-eight of them all told, with Hal Paine and Chester Crawford as their guides, were reconnoitering ten miles in advance of the main army along the river Marne in the great war between Germany and the allied armies.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure pyswarm is installed via: pip install pyswarm

# Download required NLTK resources

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """The little body of British troopers, only forty-eight of them all told, with Hal Paine

and Chester Crawford as their guides, were reconnoitering ten miles in advance of the main army along the river

Marne in the great war between Germany and the allied armies. For several hours they had been riding slowly

without encountering the enemy, when, suddenly, as the little squad topped a small hill and the two boys gained

an unobstructed view of the little plain below, Hal pulled up his horse with an exclamation."""

# Function to generate response text based on temperature

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply randomness only if temperature is below 1.0

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function to minimize (negative METEOR score)

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return -score  # Minimize negative METEOR score

# Set bounds for temperature in (0, 1]

lb = [0.01]  # Slightly above zero to avoid degenerate randomness

ub = [1.0]

# Perform PSO to find the best temperature

best\_temp, \_ = pso(objective\_function, lb, ub)

optimized\_temp = best\_temp[0]

# Generate text with optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

8)

Sample:

The speaker was an American lad of some seventeen years of age. He stopped in his walk as he spoke and grasped his companion by the arm. The latter allowed his gaze to rove over the thousands upon thousands of people who thronged the approach to the king's palace at Rome, before he replied: "Some mob, Chester; some mob."

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9741

Generated Text:

He stopped in his walk as he spoke and grasped his companion by the arm. The latter allowed his gaze to rove over the thousands upon thousands of people who thronged the approach to the king's palace at Rome, before he replied: 'Some mob, Chester; some mob.' The speaker was an American lad of some seventeen years of age.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The speaker was an American lad of some seventeen years of age. He stopped in his walk

as he spoke and grasped his companion by the arm. The latter allowed his gaze to rove over the thousands upon

thousands of people who thronged the approach to the king's palace at Rome, before he replied:

'Some mob, Chester; some mob.'

"""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment only when it's less than 1.0

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.1730, METEOR Score: 1.0000

Generated Text:

The speaker was an American lad of some seventeen years of age. He stopped in his walk as he spoke and grasped his companion by the arm. The latter allowed his gaze to rove over the thousands upon thousands of people who thronged the approach to the king's palace at Rome, before he replied: 'Some mob, Chester; some mob.'

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure pyswarm is installed via: pip install pyswarm

# Download necessary NLTK resources

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """The speaker was an American lad of some seventeen years of age. He stopped in his walk

as he spoke and grasped his companion by the arm. The latter allowed his gaze to rove over the thousands upon

thousands of people who thronged the approach to the king's palace at Rome, before he replied:

'Some mob, Chester; some mob.'"""

# Function to generate response text based on temperature

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply randomness only if temperature is below 1.0

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function to minimize (negative METEOR score)

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return -score  # Minimize negative METEOR score

# Set bounds for temperature optimization (ensuring it stays between 0 and 1)

lb = [0.01]  # Slightly above zero to avoid degenerate randomness

ub = [1.0]

# Perform PSO to find the best temperature

best\_temp, \_ = pso(objective\_function, lb, ub)

optimized\_temp = best\_temp[0]

# Generate text with optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

9)

Sample:

The two figures in the rear seat were apparently having a hard time to maintain their places, as they bounced from side to side as the car swerved first one way and then the other, or as it took a flying leap over some object in the road, which even the keen eye of the driver had failed to detect. But in spite of this, even as they bounced, they talked.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9754

Generated Text:

But in spite of this, even as they bounced, they talked. The two figures in the rear seat were apparently having a hard time to maintain their places, as they bounced from side to side as the car swerved first one way and then the other, or as it took a flying leap over some object in the road, which even the keen eye of the driver had failed to detect.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The two figures in the rear seat were apparently having a hard time to maintain their places,

as they bounced from side to side as the car swerved first one way and then the other,

or as it took a flying leap over some object in the road, which even the keen eye of the driver had failed to detect.

But in spite of this, even as they bounced, they talked."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment only when it's less than 1.0

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.2679, METEOR Score: 1.0000

Generated Text:

The two figures in the rear seat were apparently having a hard time to maintain their places, as they bounced from side to side as the car swerved first one way and then the other, or as it took a flying leap over some object in the road, which even the keen eye of the driver had failed to detect. But in spite of this, even as they bounced, they talked.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure pyswarm is installed via: pip install pyswarm

# Download necessary NLTK resources

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """The two figures in the rear seat were apparently having a hard time to maintain their places,

as they bounced from side to side as the car swerved first one way and then the other,

or as it took a flying leap over some object in the road, which even the keen eye of the driver had failed to detect.

But in spite of this, even as they bounced, they talked."""

# Function to generate response text based on temperature

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply randomness only if temperature is below 1.0

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function to minimize (negative METEOR score)

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return -score  # Minimize negative METEOR score

# Set bounds for temperature optimization (ensuring it stays between 0 and 1)

lb = [0.01]  # Slightly above zero to avoid degenerate randomness

ub = [1.0]

# Perform PSO to find the best temperature

best\_temp, \_ = pso(objective\_function, lb, ub)

optimized\_temp = best\_temp[0]

# Generate text with optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

10)

Sample:

These boys were made Rangers through the influence of Mr. Boone, and had been in the woods about a month, where they had some stirring adventures, meeting an old hermit who has helped them, and making enemies of a half-breed guide, Jean LeBlanc, and a rascally ex-deputy Ranger, Anderson by name, who was supplanted by Nate Webster, a warm-hearted old Maine guide and a firm friend of the boys.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

These boys were made Rangers through the influence of Mr. Boone, and had been in the woods about a month, where they had some stirring adventures, meeting an old hermit who has helped them, and making enemies of a half-breed guide, Jean LeBlanc, and a rascally ex-deputy Ranger, Anderson by name, who was supplanted by Nate Webster, a warm-hearted old Maine guide and a firm friend of the boys.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """These boys were made Rangers through the influence of Mr. Boone, and had been in the woods about a month,

where they had some stirring adventures, meeting an old hermit who has helped them,

and making enemies of a half-breed guide, Jean LeBlanc, and a rascally ex-deputy Ranger, Anderson by name,

who was supplanted by Nate Webster, a warm-hearted old Maine guide and a firm friend of the boys."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment only when it's less than 1.0

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.6085, METEOR Score: 1.0000

Generated Text:

These boys were made Rangers through the influence of Mr. Boone, and had been in the woods about a month, where they had some stirring adventures, meeting an old hermit who has helped them, and making enemies of a half-breed guide, Jean LeBlanc, and a rascally ex-deputy Ranger, Anderson by name, who was supplanted by Nate Webster, a warm-hearted old Maine guide and a firm friend of the boys.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you have pyswarm installed via pip: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """These boys were made Rangers through the influence of Mr. Boone, and had been in the woods about a month,

where they had some stirring adventures, meeting an old hermit who has helped them,

and making enemies of a half-breed guide, Jean LeBlanc, and a rascally ex-deputy Ranger, Anderson by name,

who was supplanted by Nate Webster, a warm-hearted old Maine guide and a firm friend of the boys."""

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment to sentence selection

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: minimize the negative METEOR score

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return -score  # We minimize the negative METEOR score to maximize it

# Bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (a slight positive number to avoid randomness issues)

ub = [1.0]   # Upper bound (maximum temperature)

# Use PSO to optimize the temperature value

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimal temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

11)

Sample:

Waupee, or the White Hawk, lived in a remote part of the forest, where animals abounded. Every day he returned from the chase with a large spoil, for he was one of the most skillful and lucky hunters of his tribe. His form was like the cedar; the fire of youth beamed from his eye; there was no forest too gloomy for him to penetrate, and no track made by bird or beast of any kind which he could not readily follow.

Manually:

Temperature: 0.5, Length: 100, METEOR Score: 0.9801

Generated Text:

Every day he returned from the chase with a large spoil, for he was one of the most skillful and lucky hunters of his tribe. His form was like the cedar; the fire of youth beamed from his eye; there was no forest too gloomy for him to penetrate, and no track made by bird or beast of any kind which he could not readily follow. Waupee, or the White Hawk, lived in a remote part of the forest, where animals abounded.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Waupee, or the White Hawk, lived in a remote part of the forest, where animals abounded.

Every day he returned from the chase with a large spoil, for he was one of the most skillful and lucky hunters of his tribe.

His form was like the cedar; the fire of youth beamed from his eye; there was no forest too gloomy for him to penetrate,

and no track made by bird or beast of any kind which he could not readily follow."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment only when it's less than 1.0

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 0.5  # Lower temperature for more creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5352, METEOR Score: 0.9947

Generated Text:

Every day he returned from the chase with a large spoil, for he was one of the most skillful and lucky hunters of his tribe. Waupee, or the White Hawk, lived in a remote part of the forest, where animals abounded. His form was like the cedar; the fire of youth beamed from his eye; there was no forest too gloomy for him to penetrate, and no track made by bird or beast of any kind which he could not readily follow.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you have pyswarm installed via pip: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Waupee, or the White Hawk, lived in a remote part of the forest, where animals abounded.

Every day he returned from the chase with a large spoil, for he was one of the most skillful and lucky hunters of his tribe.

His form was like the cedar; the fire of youth beamed from his eye; there was no forest too gloomy for him to penetrate,

and no track made by bird or beast of any kind which he could not readily follow."""

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment to sentence selection for more creativity

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: minimize the negative METEOR score

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Evaluate the METEOR score between the generated text and the original

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Penalize the generated text if it is too similar to the original

    if score > 0.5:

        score = 0.5  # Cap the score to a threshold for originality

    return -score  # Minimize negative score to maximize originality

# Bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (a slight positive number to avoid randomness issues)

ub = [1.0]   # Upper bound (maximum temperature)

# Use PSO to optimize the temperature value

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimal temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

12)

Sample:

Miss Katharine gazed vacantly about the familiar river upon whose banks she had been born and bred, and, finally noticing the sun had gone down, closing the short day, she once more drew her cloak closely about her and resumed the neglected conversation.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

Miss Katharine gazed vacantly about the familiar river upon whose banks she had been born and bred, and, finally noticing the sun had gone down, closing the short day, she once more drew her cloak closely about her and resumed the neglected conversation.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Miss Katharine gazed vacantly about the familiar river upon whose banks she had been born and bred,

and, finally noticing the sun had gone down, closing the short day, she once more drew her cloak closely about her and resumed the neglected conversation."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for variety

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.8049, METEOR Score: 1.0000

Generated Text:

Miss Katharine gazed vacantly about the familiar river upon whose banks she had been born and bred, and, finally noticing the sun had gone down, closing the short day, she once more drew her cloak closely about her and resumed the neglected conversation.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Miss Katharine gazed vacantly about the familiar river upon whose banks she had been born and bred,

and, finally noticing the sun had gone down, closing the short day, she once more drew her cloak closely about her and resumed the neglected conversation."""

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment to sentence selection for more creativity

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: minimize the negative METEOR score

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Evaluate the METEOR score between the generated text and the original

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Penalize the generated text if it is too similar to the original

    if score > 0.5:

        score = 0.5  # Cap the score to a threshold for originality

    return -score  # Minimize negative score to maximize originality

# Bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (a slight positive number to avoid randomness issues)

ub = [1.0]   # Upper bound (maximum temperature)

# Use PSO to optimize the temperature value

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimal temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

13)

Sample:

The history of the Spanish Main begins in 1509, with the voyages of Ojeda and Nicuesa, which were the first definite and authorized attempts to colonize the mainland of South America. The honor of being the first of the fifteenth-century navigators to set foot upon either of the two American continents, indisputably belongs to John Cabot, on June 24, 1497. Who was next to make a continental landfall, and in the more southerly latitudes, is a question which lies between Columbus and Amerigo Vespucci.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

The history of the Spanish Main begins in 1509, with the voyages of Ojeda and Nicuesa, which were the first definite and authorized attempts to colonize the mainland of South America. The honor of being the first of the fifteenth-century navigators to set foot upon either of the two American continents, indisputably belongs to John Cabot, on June 24, 1497. Who was next to make a continental landfall, and in the more southerly latitudes, is a question which lies between Columbus and Amerigo Vespucci.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The history of the Spanish Main begins in 1509, with the voyages of Ojeda and Nicuesa,

which were the first definite and authorized attempts to colonize the mainland of South America.

The honor of being the first of the fifteenth-century navigators to set foot upon either of the two American continents,

indisputably belongs to John Cabot, on June 24, 1497.

Who was next to make a continental landfall, and in the more southerly latitudes, is a question which lies between Columbus and Amerigo Vespucci."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for variety

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.1801, METEOR Score: 0.8814

Generated Text:

The honor of being the first of the fifteenth-century navigators to set foot upon either of the two American continents, indisputably belongs to John Cabot, on June 24, 1497. Who was next to make a continental landfall, and in the more southerly latitudes, is a question which lies between Columbus and Amerigo Vespucci. The history of the Spanish Main begins in 1509, with the voyages of Ojeda and Nicuesa, which were the first definite and authorized attempts to colonize the mainland of South America.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you have pyswarm installed via pip: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The history of the Spanish Main begins in 1509, with the voyages of Ojeda and Nicuesa,

which were the first definite and authorized attempts to colonize the mainland of South America.

The honor of being the first of the fifteenth-century navigators to set foot upon either of the two American continents,

indisputably belongs to John Cabot, on June 24, 1497.

Who was next to make a continental landfall, and in the more southerly latitudes, is a question which lies between Columbus and Amerigo Vespucci."""

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment to sentence selection for more creativity

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: minimize the negative METEOR score

def objective\_function(temp):

    temperature = temp[0]

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Evaluate the METEOR score between the generated text and the original

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Penalize the generated text if it is too similar to the original

    if score > 0.5:

        score = 0.5  # Cap the score to a threshold for originality

    return score  # Minimize the METEOR score (lower is better)

# Bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (a slight positive number to avoid randomness issues)

ub = [1.0]   # Upper bound (maximum temperature)

# Use PSO to optimize the temperature value

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimal temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

14)

Sample:

The guard is always relieved at the appointed intervals with military formality and precision. One soldier, older, taller than the rest, is in command of the other four. From his buttonhole dangles from a white ribbon a little cross of white enamel. Though he shows no insignia of rank higher than that of a Sergeant of the Guard, he has won the proud distinction of the Legion of Honor.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9453

Generated Text:

Though he shows no insignia of rank higher than that of a Sergeant of the Guard, he has won the proud distinction of the Legion of Honor. The guard is always relieved at the appointed intervals with military formality and precision. One soldier, older, taller than the rest, is in command of the other four. From his buttonhole dangles from a white ribbon a little cross of white enamel.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The guard is always relieved at the appointed intervals with military formality and precision.

One soldier, older, taller than the rest, is in command of the other four.

From his buttonhole dangles from a white ribbon a little cross of white enamel.

Though he shows no insignia of rank higher than that of a Sergeant of the Guard,

he has won the proud distinction of the Legion of Honor."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for variety

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.2066, METEOR Score: 0.9290

Generated Text:

Though he shows no insignia of rank higher than that of a Sergeant of the Guard, he has won the proud distinction of the Legion of Honor. The guard is always relieved at the appointed intervals with military formality and precision. From his buttonhole dangles from a white ribbon a little cross of white enamel. One soldier, older, taller than the rest, is in command of the other four.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The guard is always relieved at the appointed intervals with military formality and precision.

One soldier, older, taller than the rest, is in command of the other four.

From his buttonhole dangles from a white ribbon a little cross of white enamel.

Though he shows no insignia of rank higher than that of a Sergeant of the Guard,

he has won the proud distinction of the Legion of Honor."""

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (lower temperature gives more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

15)

Sample:

Came Christmas Eve. They had no money and no food and no fire. Stop! The fire of love burned in the woman’s heart, the fire of hate in the man’s. Prison life usually completes the education in shame of the unfortunate men who are thrust there. This was before the days in which humane men interested themselves in prisons and prisoners and strove to awaken the world to its responsibilities to, as well as the possibilities of, the convict.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9999

Generated Text:

The fire of love burned in the woman’s heart, the fire of hate in the man’s. Prison life usually completes the education in shame of the unfortunate men who are thrust there. Stop! They had no money and no food and no fire. Came Christmas Eve. This was before the days in which humane men interested themselves in prisons and prisoners and strove to awaken the world to its responsibilities to, as well as the possibilities of, the convict.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Came Christmas Eve. They had no money and no food and no fire. Stop!

The fire of love burned in the woman’s heart, the fire of hate in the man’s.

Prison life usually completes the education in shame of the unfortunate men who are thrust there.

This was before the days in which humane men interested themselves in prisons and prisoners and

strove to awaken the world to its responsibilities to, as well as the possibilities of, the convict."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for variety

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.7833, METEOR Score: 0.9194

Generated Text:

Prison life usually completes the education in shame of the unfortunate men who are thrust there. This was before the days in which humane men interested themselves in prisons and prisoners and strove to awaken the world to its responsibilities to, as well as the possibilities of, the convict. The fire of love burned in the woman’s heart, the fire of hate in the man’s. Came Christmas Eve. They had no money and no food and no fire. Stop!

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Came Christmas Eve. They had no money and no food and no fire. Stop!

The fire of love burned in the woman’s heart, the fire of hate in the man’s.

Prison life usually completes the education in shame of the unfortunate men who are thrust there.

This was before the days in which humane men interested themselves in prisons and prisoners and

strove to awaken the world to its responsibilities to, as well as the possibilities of, the convict."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (lower temperature gives more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

16)

Sample:

Elizabeth Ann’s Great-aunt Harriet was a widow who was not very rich or very poor, and she had one daughter, Frances, who gave piano lessons to little girls. They kept a “girl” whose name was Grace and who had asthma dreadfully and wasn’t very much of a “girl” at all, being nearer fifty than forty. Aunt Harriet, who was very tender-hearted, kept her chiefly because she couldn’t get any other place on account of her coughing so you could hear her all over the house.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9553

Generated Text:

They kept a “girl” whose name was Grace and who had asthma dreadfully and wasn’t very much of a “girl” at all, being nearer fifty than forty. Aunt Harriet, who was very tender-hearted, kept her chiefly because she couldn’t get any other place on account of her coughing so you could hear her all over the house. Elizabeth Ann’s Great-aunt Harriet was a widow who was not very rich or very poor, and she had one daughter, Frances, who gave piano lessons to little girls.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Elizabeth Ann’s Great-aunt Harriet was a widow who was not very rich or very poor,

and she had one daughter, Frances, who gave piano lessons to little girls. They kept a “girl”

whose name was Grace and who had asthma dreadfully and wasn’t very much of a “girl” at all, being

nearer fifty than forty. Aunt Harriet, who was very tender-hearted, kept her chiefly because she

couldn’t get any other place on account of her coughing so you could hear her all over the house."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for variety

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.2037, METEOR Score: 0.9707

Generated Text:

They kept a “girl” whose name was Grace and who had asthma dreadfully and wasn’t very much of a “girl” at all, being nearer fifty than forty. Elizabeth Ann’s Great-aunt Harriet was a widow who was not very rich or very poor, and she had one daughter, Frances, who gave piano lessons to little girls. Aunt Harriet, who was very tender-hearted, kept her chiefly because she couldn’t get any other place on account of her coughing so you could hear her all over the house.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Elizabeth Ann’s Great-aunt Harriet was a widow who was not very rich or very poor,

and she had one daughter, Frances, who gave piano lessons to little girls. They kept a “girl”

whose name was Grace and who had asthma dreadfully and wasn’t very much of a “girl” at all, being

nearer fifty than forty. Aunt Harriet, who was very tender-hearted, kept her chiefly because she

couldn’t get any other place on account of her coughing so you could hear her all over the house."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

17)

Sample:

He happened to be building a Palace when the news came, and he left all the bricks kicking about the floor for Nurse to clear up—but then the news was rather remarkable news. You see, there was a knock at the front door and voices talking downstairs, and Lionel thought it was the man come to see about the gas, which had not been allowed to be lighted since the day when Lionel made a swing by tying his skipping rope to the gas bracket. And then, quite suddenly, Nurse came in and said, "Master Lionel, dear, they've come to fetch you to go and be King."

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9920

Generated Text:

He happened to be building a Palace when the news came, and he left all the bricks kicking about the floor for Nurse to clear up—but then the news was rather remarkable news. And then, quite suddenly, Nurse came in and said, "Master Lionel, dear, they've come to fetch you to go and be King. You see, there was a knock at the front door and voices talking downstairs, and Lionel thought it was the man come to see about the gas, which had not been allowed to be lighted since the day when Lionel made a swing by tying his skipping rope to the gas bracket.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """He happened to be building a Palace when the news came, and he left all the bricks kicking

about the floor for Nurse to clear up—but then the news was rather remarkable news. You see, there was a knock

at the front door and voices talking downstairs, and Lionel thought it was the man come to see about the gas,

which had not been allowed to be lighted since the day when Lionel made a swing by tying his skipping rope to

the gas bracket. And then, quite suddenly, Nurse came in and said, "Master Lionel, dear, they've come to fetch

you to go and be King."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for variety

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.1542, METEOR Score: 0.8879

Generated Text:

You see, there was a knock at the front door and voices talking downstairs, and Lionel thought it was the man come to see about the gas, which had not been allowed to be lighted since the day when Lionel made a swing by tying his skipping rope to the gas bracket. And then, quite suddenly, Nurse came in and said, "Master Lionel, dear, they've come to fetch you to go and be King. He happened to be building a Palace when the news came, and he left all the bricks kicking about the floor for Nurse to clear up—but then the news was rather remarkable news.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """He happened to be building a Palace when the news came, and he left all the bricks kicking

about the floor for Nurse to clear up—but then the news was rather remarkable news. You see, there was a knock

at the front door and voices talking downstairs, and Lionel thought it was the man come to see about the gas,

which had not been allowed to be lighted since the day when Lionel made a swing by tying his skipping rope to

the gas bracket. And then, quite suddenly, Nurse came in and said, "Master Lionel, dear, they've come to fetch

you to go and be King."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

18)

Sample:

We are the Bastables—Oswald, Dora, Dicky, Alice, Noel, and H. O. If you want to know why we call our youngest brother H. O. you can jolly well read The Treasure Seekers and find out. We were the Treasure Seekers, and we sought it high and low, and quite regularly, because we particularly wanted to find it. And at last we did not find it, but we were found by a good, kind Indian uncle, who helped Father with his business, so that Father was able to take us all to live in a jolly big red house on Blackheath, instead of in the Lewisham Road, where we lived when we were only poor but honest Treasure Seekers.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

We are the Bastables—Oswald, Dora, Dicky, Alice, Noel, and H. O. If you want to know why we call our youngest brother H. O. you can jolly well read The Treasure Seekers and find out. We were the Treasure Seekers, and we sought it high and low, and quite regularly, because we particularly wanted to find it. And at last we did not find it, but we were found by a good, kind Indian uncle, who helped Father with his business, so that Father was able to take us all to live in a jolly big red house on Blackheath, instead of in the Lewisham Road, where we lived when we were only poor but honest Treasure Seekers.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """We are the Bastables—Oswald, Dora, Dicky, Alice, Noel, and H. O. If you want to know why we

call our youngest brother H. O. you can jolly well read The Treasure Seekers and find out. We were the

Treasure Seekers, and we sought it high and low, and quite regularly, because we particularly wanted to find

it. And at last we did not find it, but we were found by a good, kind Indian uncle, who helped Father with

his business, so that Father was able to take us all to live in a jolly big red house on Blackheath, instead

of in the Lewisham Road, where we lived when we were only poor but honest Treasure Seekers."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for variety

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.2478, METEOR Score: 0.9926

Generated Text:

If you want to know why we call our youngest brother H. O. you can jolly well read The Treasure Seekers and find out. We were the Treasure Seekers, and we sought it high and low, and quite regularly, because we particularly wanted to find it. We are the Bastables—Oswald, Dora, Dicky, Alice, Noel, and H. O. And at last we did not find it, but we were found by a good, kind Indian uncle, who helped Father with his business, so that Father was able to take us all to live in a jolly big red house on Blackheath, instead of in the Lewisham Road, where we lived when we were only poor but honest Treasure Seekers.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """We are the Bastables—Oswald, Dora, Dicky, Alice, Noel, and H. O. If you want to know why we

call our youngest brother H. O. you can jolly well read The Treasure Seekers and find out. We were the

Treasure Seekers, and we sought it high and low, and quite regularly, because we particularly wanted to find

it. And at last we did not find it, but we were found by a good, kind Indian uncle, who helped Father with

his business, so that Father was able to take us all to live in a jolly big red house on Blackheath, instead

of in the Lewisham Road, where we lived when we were only poor but honest Treasure Seekers."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

19)

Sample:

The White House was on the edge of a hill, with a wood behind it - and the chalk-quarry on one side and the gravel-pit on the other. Down at the bottom of the hill was a level plain, with queer-shaped white buildings where people burnt lime, and a big red brewery and other houses; and when the big chimneys were smoking and the sun was setting, the valley looked as if it was filled with golden mist, and the limekilns and oast-houses glimmered and glittered till they were like an enchanted city out of the Arabian Nights.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.8473

Generated Text:

Down at the bottom of the hill was a level plain, with queer-shaped white buildings where people burnt lime, and a big red brewery and other houses; and when the big chimneys were smoking and the sun was setting, the valley looked as if it was filled with golden mist, and the limekilns and oast-houses glimmered and glittered till they were like an enchanted city out of the Arabian Nights. The White House was on the edge of a hill, with a wood behind it - and the chalk-quarry on one side and the gravel-pit on the other.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The White House was on the edge of a hill, with a wood behind it - and the chalk-quarry

on one side and the gravel-pit on the other. Down at the bottom of the hill was a level plain, with

queer-shaped white buildings where people burnt lime, and a big red brewery and other houses; and when

the big chimneys were smoking and the sun was setting, the valley looked as if it was filled with golden

mist, and the limekilns and oast-houses glimmered and glittered till they were like an enchanted city out

of the Arabian Nights."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.4088, METEOR Score: 1.0000

Generated Text:

The White House was on the edge of a hill, with a wood behind it - and the chalk-quarry on one side and the gravel-pit on the other. Down at the bottom of the hill was a level plain, with queer-shaped white buildings where people burnt lime, and a big red brewery and other houses; and when the big chimneys were smoking and the sun was setting, the valley looked as if it was filled with golden mist, and the limekilns and oast-houses glimmered and glittered till they were like an enchanted city out of the Arabian Nights.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """The White House was on the edge of a hill, with a wood behind it - and the chalk-quarry

on one side and the gravel-pit on the other. Down at the bottom of the hill was a level plain, with

queer-shaped white buildings where people burnt lime, and a big red brewery and other houses; and when

the big chimneys were smoking and the sun was setting, the valley looked as if it was filled with golden

mist, and the limekilns and oast-houses glimmered and glittered till they were like an enchanted city out

of the Arabian Nights."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

20)

Sample:

There were once four children who spent their summer holidays in a white house, happily situated between a sandpit and a chalkpit. One day they had the good fortune to find in the sandpit a strange creature. Its eyes were on long horns like snail’s eyes, and it could move them in and out like telescopes. It had ears like a bat’s ears, and its tubby body was shaped like a spider’s and covered with thick soft fur—and it had hands and feet like a monkey’s.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9375

Generated Text:

One day they had the good fortune to find in the sandpit a strange creature. There were once four children who spent their summer holidays in a white house, happily situated between a sandpit and a chalkpit. It had ears like a bat’s ears, and its tubby body was shaped like a spider’s and covered with thick soft fur—and it had hands and feet like a monkey’s. Its eyes were on long horns like snail’s eyes, and it could move them in and out like telescopes.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """There were once four children who spent their summer holidays in a white house,

happily situated between a sandpit and a chalkpit. One day they had the good fortune to find in

the sandpit a strange creature. Its eyes were on long horns like snail’s eyes, and it could move

them in and out like telescopes. It had ears like a bat’s ears, and its tubby body was shaped like

a spider’s and covered with thick soft fur—and it had hands and feet like a monkey’s."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.6751, METEOR Score: 0.9458

Generated Text:

Its eyes were on long horns like snail’s eyes, and it could move them in and out like telescopes. It had ears like a bat’s ears, and its tubby body was shaped like a spider’s and covered with thick soft fur—and it had hands and feet like a monkey’s. There were once four children who spent their summer holidays in a white house, happily situated between a sandpit and a chalkpit. One day they had the good fortune to find in the sandpit a strange creature.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Make sure to install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """There were once four children who spent their summer holidays in a white house,

happily situated between a sandpit and a chalkpit. One day they had the good fortune to find in

the sandpit a strange creature. Its eyes were on long horns like snail’s eyes, and it could move

them in and out like telescopes. It had ears like a bat’s ears, and its tubby body was shaped like

a spider’s and covered with thick soft fur—and it had hands and feet like a monkey’s."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

21)

Sample:

They were not railway children to begin with. I don't suppose they had ever thought about railways except as a means of getting to Maskelyne and Cook's, the Pantomime, Zoological Gardens, and Madame Tussaud's. They were just ordinary suburban children, and they lived with their Father and Mother in an ordinary red-brick-fronted villa, with coloured glass in the front door, a tiled passage that was called a hall, a bath-room with hot and cold water, electric bells, French windows, and a good deal of white paint, and 'every modern convenience', as the house-agents say.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9554

Generated Text:

They were just ordinary suburban children, and they lived with their Father and Mother in an ordinary red-brick-fronted villa, with coloured glass in the front door, a tiled passage that was called a hall, a bath-room with hot and cold water, electric bells, French windows, and a good deal of white paint, and 'every modern convenience', as the house-agents say. They were not railway children to begin with. I don't suppose they had ever thought about railways except as a means of getting to Maskelyne and Cook's, the Pantomime, Zoological Gardens, and Madame Tussaud's.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """They were not railway children to begin with. I don't suppose they had ever thought

about railways except as a means of getting to Maskelyne and Cook's, the Pantomime, Zoological Gardens,

and Madame Tussaud's. They were just ordinary suburban children, and they lived with their Father and Mother

in an ordinary red-brick-fronted villa, with coloured glass in the front door, a tiled passage that was called

a hall, a bath-room with hot and cold water, electric bells, French windows, and a good deal of white paint,

and 'every modern convenience', as the house-agents say."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5696, METEOR Score: 0.9334

Generated Text:

They were just ordinary suburban children, and they lived with their Father and Mother in an ordinary red-brick-fronted villa, with coloured glass in the front door, a tiled passage that was called a hall, a bath-room with hot and cold water, electric bells, French windows, and a good deal of white paint, and 'every modern convenience', as the house-agents say. I don't suppose they had ever thought about railways except as a means of getting to Maskelyne and Cook's, the Pantomime, Zoological Gardens, and Madame Tussaud's. They were not railway children to begin with.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you install the pyswarm library (pip install pyswarm)

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """They were not railway children to begin with. I don't suppose they had ever thought

about railways except as a means of getting to Maskelyne and Cook's, the Pantomime, Zoological Gardens,

and Madame Tussaud's. They were just ordinary suburban children, and they lived with their Father and Mother

in an ordinary red-brick-fronted villa, with coloured glass in the front door, a tiled passage that was called

a hall, a bath-room with hot and cold water, electric bells, French windows, and a good deal of white paint,

and 'every modern convenience', as the house-agents say."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

22)

Sample:

Philip Haldane and his sister lived in a little red-roofed house in a little red-roofed town. They had a little garden and a little balcony, and a little stable with a little pony in it—and a little cart for the pony to draw; a little canary hung in a little cage in the little bow-window, and the neat little servant kept everything as bright and clean as a little new pin.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.8965

Generated Text:

They had a little garden and a little balcony, and a little stable with a little pony in it—and a little cart for the pony to draw; a little canary hung in a little cage in the little bow-window, and the neat little servant kept everything as bright and clean as a little new pin. Philip Haldane and his sister lived in a little red-roofed house in a little red-roofed town.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Philip Haldane and his sister lived in a little red-roofed house in a little red-roofed town.

They had a little garden and a little balcony, and a little stable with a little pony in it—and a little cart

for the pony to draw; a little canary hung in a little cage in the little bow-window, and the neat little

servant kept everything as bright and clean as a little new pin."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.6445, METEOR Score: 0.8965

Generated Text:

They had a little garden and a little balcony, and a little stable with a little pony in it—and a little cart for the pony to draw; a little canary hung in a little cage in the little bow-window, and the neat little servant kept everything as bright and clean as a little new pin. Philip Haldane and his sister lived in a little red-roofed house in a little red-roofed town.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Make sure to install pyswarm using pip

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Philip Haldane and his sister lived in a little red-roofed house in a little red-roofed town.

They had a little garden and a little balcony, and a little stable with a little pony in it—and a little cart

for the pony to draw; a little canary hung in a little cage in the little bow-window, and the neat little

servant kept everything as bright and clean as a little new pin."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

23)

Sample:

Jane gave the letter an affectionate little pat. It was almost as though she had heard lively little Adrienne's voice. How good it was, she reflected happily, to know that this time she would go East, not as a lonely outlander, but as one whose place awaited her. There would be smiling faces and welcoming hands to greet her when she climbed the steps of Madison Hall. Yes, Wellington was truly her Alma Mater and Madison Hall her second home.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9847

Generated Text:

It was almost as though she had heard lively little Adrienne's voice. There would be smiling faces and welcoming hands to greet her when she climbed the steps of Madison Hall. Yes, Wellington was truly her Alma Mater and Madison Hall her second home. Jane gave the letter an affectionate little pat. How good it was, she reflected happily, to know that this time she would go East, not as a lonely outlander, but as one whose place awaited her.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Jane gave the letter an affectionate little pat. It was almost as though she had heard lively little Adrienne's voice.

How good it was, she reflected happily, to know that this time she would go East, not as a lonely outlander, but as one whose place awaited her.

There would be smiling faces and welcoming hands to greet her when she climbed the steps of Madison Hall. Yes, Wellington was truly her Alma Mater

and Madison Hall her second home."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.9708, METEOR Score: 0.9933

Generated Text:

How good it was, she reflected happily, to know that this time she would go East, not as a lonely outlander, but as one whose place awaited her. Jane gave the letter an affectionate little pat. It was almost as though she had heard lively little Adrienne's voice. There would be smiling faces and welcoming hands to greet her when she climbed the steps of Madison Hall. Yes, Wellington was truly her Alma Mater and Madison Hall her second home.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Make sure to install pyswarm using pip

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Jane gave the letter an affectionate little pat. It was almost as though she had heard lively little Adrienne's voice.

How good it was, she reflected happily, to know that this time she would go East, not as a lonely outlander, but as one whose place awaited her.

There would be smiling faces and welcoming hands to greet her when she climbed the steps of Madison Hall. Yes, Wellington was truly her Alma Mater

and Madison Hall her second home."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if 0 < temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

24)

Sample:

It was seldom that this young girl appeared anywhere in public without attracting the attention of any who chanced to glance into her sweet face. Its contour was almost perfect and the coloring exquisite. In addition she had a slender form which she carried with exceeding grace and a modest, winning demeanor that was more demure and unconscious than shy.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9949

Generated Text:

It was seldom that this young girl appeared anywhere in public without attracting the attention of any who chanced to glance into her sweet face. In addition she had a slender form which she carried with exceeding grace and a modest, winning demeanor that was more demure and unconscious than shy. Its contour was almost perfect and the coloring exquisite.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """It was seldom that this young girl appeared anywhere in public without attracting the attention of any who chanced

to glance into her sweet face. Its contour was almost perfect and the coloring exquisite. In addition she had a slender form which

she carried with exceeding grace and a modest, winning demeanor that was more demure and unconscious than shy."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.6885, METEOR Score: 0.9754

Generated Text:

In addition she had a slender form which she carried with exceeding grace and a modest, winning demeanor that was more demure and unconscious than shy. Its contour was almost perfect and the coloring exquisite. It was seldom that this young girl appeared anywhere in public without attracting the attention of any who chanced to glance into her sweet face.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Make sure to install pyswarm using pip

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """It was seldom that this young girl appeared anywhere in public without attracting the attention of any who chanced

to glance into her sweet face. Its contour was almost perfect and the coloring exquisite. In addition she had a slender form which

she carried with exceeding grace and a modest, winning demeanor that was more demure and unconscious than shy."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

25)

Sample:

The girl they were discussing came leisurely down a path, her books under one arm, the other hand holding a class paper which she examined in a cursory way as she walked. She wore a dark skirt and a simple shirtwaist, both quite modish and becoming, and her shoes were the admiration and envy of half the girls at the school. Dorothy Knerr used to say that "Mary Louise's clothes always looked as if they grew on her," but that may have been partially accounted for by the grace of her slim form and her unconscious but distinctive poise of bearing. Few people would describe Mary Louise Burrows as beautiful, while all would agree that she possessed charming manners. And she was fifteen—an age when many girls are both awkward and shy.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9222

Generated Text:

And she was fifteen—an age when many girls are both awkward and shy. Few people would describe Mary Louise Burrows as beautiful, while all would agree that she possessed charming manners. She wore a dark skirt and a simple shirtwaist, both quite modish and becoming, and her shoes were the admiration and envy of half the girls at the school. Dorothy Knerr used to say that "Mary Louise's clothes always looked as if they grew on her," but that may have been partially accounted for by the grace of her slim form and her unconscious but distinctive poise of bearing. The girl they were discussing came leisurely down a path, her books under one arm, the other hand holding a class paper which she examined in a cursory way as she walked.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The girl they were discussing came leisurely down a path, her books under one arm, the other hand holding a

class paper which she examined in a cursory way as she walked. She wore a dark skirt and a simple shirtwaist, both quite modish

and becoming, and her shoes were the admiration and envy of half the girls at the school. Dorothy Knerr used to say that

"Mary Louise's clothes always looked as if they grew on her," but that may have been partially accounted for by the grace of

her slim form and her unconscious but distinctive poise of bearing. Few people would describe Mary Louise Burrows as beautiful,

while all would agree that she possessed charming manners. And she was fifteen—an age when many girls are both awkward and shy."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.2009, METEOR Score: 0.9597

Generated Text:

She wore a dark skirt and a simple shirtwaist, both quite modish and becoming, and her shoes were the admiration and envy of half the girls at the school. Few people would describe Mary Louise Burrows as beautiful, while all would agree that she possessed charming manners. Dorothy Knerr used to say that "Mary Louise's clothes always looked as if they grew on her," but that may have been partially accounted for by the grace of her slim form and her unconscious but distinctive poise of bearing. And she was fifteen—an age when many girls are both awkward and shy. The girl they were discussing came leisurely down a path, her books under one arm, the other hand holding a class paper which she examined in a cursory way as she walked.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install using: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The girl they were discussing came leisurely down a path, her books under one arm, the other hand holding a

class paper which she examined in a cursory way as she walked. She wore a dark skirt and a simple shirtwaist, both quite modish

and becoming, and her shoes were the admiration and envy of half the girls at the school. Dorothy Knerr used to say that

"Mary Louise's clothes always looked as if they grew on her," but that may have been partially accounted for by the grace of

her slim form and her unconscious but distinctive poise of bearing. Few people would describe Mary Louise Burrows as beautiful,

while all would agree that she possessed charming manners. And she was fifteen—an age when many girls are both awkward and shy."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

26)

Sample:

The Princess Ailinn was over head and ears in love with Cuglas, and Cuglas was over head and ears in love with the Princess Ailinn, and he believed that never was summer morning half as bright, or as sweet, or as fair as she. The glimpse which he had just caught of her filled his heart with delight, and almost put all thought of hunting out of his head, when suddenly the tuneful cries of the hounds, answered by a hundred echoes from the groves, broke upon his ear.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

The Princess Ailinn was over head and ears in love with Cuglas, and Cuglas was over head and ears in love with the Princess Ailinn, and he believed that never was summer morning half as bright, or as sweet, or as fair as she. The glimpse which he had just caught of her filled his heart with delight, and almost put all thought of hunting out of his head, when suddenly the tuneful cries of the hounds, answered by a hundred echoes from the groves, broke upon his ear.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The Princess Ailinn was over head and ears in love with Cuglas, and Cuglas was over head and ears in love with

the Princess Ailinn, and he believed that never was summer morning half as bright, or as sweet, or as fair as she. The glimpse

which he had just caught of her filled his heart with delight, and almost put all thought of hunting out of his head, when

suddenly the tuneful cries of the hounds, answered by a hundred echoes from the groves, broke upon his ear."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5240, METEOR Score: 0.9809

Generated Text:

The glimpse which he had just caught of her filled his heart with delight, and almost put all thought of hunting out of his head, when suddenly the tuneful cries of the hounds, answered by a hundred echoes from the groves, broke upon his ear. The Princess Ailinn was over head and ears in love with Cuglas, and Cuglas was over head and ears in love with the Princess Ailinn, and he believed that never was summer morning half as bright, or as sweet, or as fair as she.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install using: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The Princess Ailinn was over head and ears in love with Cuglas, and Cuglas was over head and ears in love with

the Princess Ailinn, and he believed that never was summer morning half as bright, or as sweet, or as fair as she. The glimpse

which he had just caught of her filled his heart with delight, and almost put all thought of hunting out of his head, when

suddenly the tuneful cries of the hounds, answered by a hundred echoes from the groves, broke upon his ear."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (close to zero, but positive)

ub = [1.0]   # Upper bound (temperature can't exceed 1)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

27)

Sample:

Master Penrose always gave a full holiday on Saturday. Then the wharves were sure to swarm with the mischievous little chaps, all eager to carry out some favorite plan for amusement, in which old Ocean was sure to be engaged as a play-fellow. Poor indeed was the lad who had not a fish-hook and line with which to try his skill. The very youngest had his tiny boat to be launched, while his elders were planning sailing-parties, or jumping and leaping in the water like so many dolphins.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9375

Generated Text:

Poor indeed was the lad who had not a fish-hook and line with which to try his skill. Master Penrose always gave a full holiday on Saturday. The very youngest had his tiny boat to be launched, while his elders were planning sailing-parties, or jumping and leaping in the water like so many dolphins. Then the wharves were sure to swarm with the mischievous little chaps, all eager to carry out some favorite plan for amusement, in which old Ocean was sure to be engaged as a play-fellow.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Master Penrose always gave a full holiday on Saturday. Then the wharves were sure to swarm with the mischievous little chaps,

all eager to carry out some favorite plan for amusement, in which old Ocean was sure to be engaged as a play-fellow. Poor indeed was the lad

who had not a fish-hook and line with which to try his skill. The very youngest had his tiny boat to be launched, while his elders were

planning sailing-parties, or jumping and leaping in the water like so many dolphins."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.0807, METEOR Score: 0.9997

Generated Text:

Then the wharves were sure to swarm with the mischievous little chaps, all eager to carry out some favorite plan for amusement, in which old Ocean was sure to be engaged as a play-fellow. Master Penrose always gave a full holiday on Saturday. Poor indeed was the lad who had not a fish-hook and line with which to try his skill. The very youngest had his tiny boat to be launched, while his elders were planning sailing-parties, or jumping and leaping in the water like so many dolphins.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install using: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Master Penrose always gave a full holiday on Saturday. Then the wharves were sure to swarm with the mischievous little chaps,

all eager to carry out some favorite plan for amusement, in which old Ocean was sure to be engaged as a play-fellow. Poor indeed was the lad

who had not a fish-hook and line with which to try his skill. The very youngest had his tiny boat to be launched, while his elders were

planning sailing-parties, or jumping and leaping in the water like so many dolphins."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO: Minimize the METEOR score

def objective\_function(temp):

    temperature = temp[0]  # The temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute the METEOR score between the generated and original text

    score = meteor\_score([original\_text.split()], generated\_tokens)

    # Return the METEOR score to be minimized

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Lower bound (avoids zero, ensures some randomness)

ub = [1.0]   # Upper bound (full randomness)

# Use PSO to optimize the temperature value that minimizes the METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# The optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute METEOR score for the generated text with optimized temperature

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print the results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

28)

Sample:

At this instant, the feet of the steam man began rising and falling with lightning like rapidity, the wagon being jerked forward with such sudden swiftness, that both Ethan and Mickey turned back summersets, rolling heels over head off the vehicle to the ground, while the monster went puffing over the prairie, and at a terrific rate. Baldy was about to start in pursuit of it, when Johnny, the deformed boy, restrained him.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9807

Generated Text:

Baldy was about to start in pursuit of it, when Johnny, the deformed boy, restrained him. At this instant, the feet of the steam man began rising and falling with lightning-like rapidity, the wagon being jerked forward with such sudden swiftness, that both Ethan and Mickey turned back summersaults, rolling heels over head off the vehicle to the ground, while the monster went puffing over the prairie at a terrific rate.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """At this instant, the feet of the steam man began rising and falling with lightning-like rapidity, the wagon being jerked

forward with such sudden swiftness, that both Ethan and Mickey turned back summersaults, rolling heels over head off the vehicle to the

ground, while the monster went puffing over the prairie at a terrific rate. Baldy was about to start in pursuit of it, when Johnny, the

deformed boy, restrained him."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.8470, METEOR Score: 1.0000

Generated Text:

At this instant, the feet of the steam man began rising and falling with lightning-like rapidity, the wagon being jerked forward with such sudden swiftness, that both Ethan and Mickey turned back summersaults, rolling heels over head off the vehicle to the ground, while the monster went puffing over the prairie at a terrific rate. Baldy was about to start in pursuit of it, when Johnny, the deformed boy, restrained him.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install using: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """At this instant, the feet of the steam man began rising and falling with lightning-like rapidity, the wagon being jerked

forward with such sudden swiftness, that both Ethan and Mickey turned back summersaults, rolling heels over head off the vehicle to the

ground, while the monster went puffing over the prairie at a terrific rate. Baldy was about to start in pursuit of it, when Johnny, the

deformed boy, restrained him."""

# Function to generate text with temperature optimization

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature scaling (higher randomness for lower temperature)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Ensure at least one sentence is selected

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract single temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero for some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

29)

Sample:

One beautiful misummer night in 18— a large, heavily laden steamer was making her way swiftly up the Pacific coast, in the direction of San Francisco. She was opposite the California shore, only a day's sail distant from the City of the Golden Gate, and many of the passengers had already begun making preparations for landing, even though a whole night and the better part of a day was to intervene ere they could expect to set their feet upon solid land.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9328

Generated Text:

She was opposite the California shore, only a day's sail distant from the City of the Golden Gate, and many of the passengers had already begun making preparations for landing, even though a whole night and the better part of a day was to intervene ere they could expect to set their feet upon solid land. One beautiful midsummer night in 18— a large, heavily laden steamer was making her way swiftly up the Pacific coast, in the direction of San Francisco.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """One beautiful midsummer night in 18— a large, heavily laden steamer was making her way swiftly up the Pacific coast,

in the direction of San Francisco. She was opposite the California shore, only a day's sail distant from the City of the Golden Gate,

and many of the passengers had already begun making preparations for landing, even though a whole night and the better part of a day

was to intervene ere they could expect to set their feet upon solid land."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5570, METEOR Score: 0.9328

Generated Text:

She was opposite the California shore, only a day's sail distant from the City of the Golden Gate, and many of the passengers had already begun making preparations for landing, even though a whole night and the better part of a day was to intervene ere they could expect to set their feet upon solid land. One beautiful midsummer night in 18— a large, heavily laden steamer was making her way swiftly up the Pacific coast, in the direction of San Francisco.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install using: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """One beautiful midsummer night in 18— a large, heavily laden steamer was making her way swiftly up the Pacific coast,

in the direction of San Francisco. She was opposite the California shore, only a day's sail distant from the City of the Golden Gate,

and many of the passengers had already begun making preparations for landing, even though a whole night and the better part of a day

was to intervene ere they could expect to set their feet upon solid land."""

# Function to generate text with temperature optimization

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature scaling (higher randomness for lower temperature)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Ensure at least one sentence is selected

    num\_sentences = max(1, response\_length // 10)

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract single temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero for some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

30)

Sample:

Nick, although born in western Pennsylvania, was as thoroughly Dutch as if he had first opened his eyes on the banks of the Zuyder Zee, in the lowlands of Holland. His parents had come from that part of the world which has produced so many fine scholars and done so much for science and literature. They talked the language of the Fatherland, although they occasionally ventured on very broken English for the instruction of the boy and girl which heaven had given them.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9713

Generated Text:

Nick, although born in western Pennsylvania, was as thoroughly Dutch as if he had first opened his eyes on the banks of the Zuyder Zee, in the lowlands of Holland. They talked the language of the Fatherland, although they occasionally ventured on very broken English for the instruction of the boy and girl which heaven had given them. His parents had come from that part of the world which has produced so many fine scholars and done so much for science and literature.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Nick, although born in western Pennsylvania, was as thoroughly Dutch as if he had first opened his eyes on the banks of

the Zuyder Zee, in the lowlands of Holland. His parents had come from that part of the world which has produced so many fine scholars

and done so much for science and literature. They talked the language of the Fatherland, although they occasionally ventured on very

broken English for the instruction of the boy and girl which heaven had given them."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5218, METEOR Score: 1.0000

Generated Text:

Nick, although born in western Pennsylvania, was as thoroughly Dutch as if he had first opened his eyes on the banks of the Zuyder Zee, in the lowlands of Holland. His parents had come from that part of the world which has produced so many fine scholars and done so much for science and literature. They talked the language of the Fatherland, although they occasionally ventured on very broken English for the instruction of the boy and girl which heaven had given them.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install with: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Nick, although born in western Pennsylvania, was as thoroughly Dutch as if he had first opened his eyes on the banks of

the Zuyder Zee, in the lowlands of Holland. His parents had come from that part of the world which has produced so many fine scholars

and done so much for science and literature. They talked the language of the Fatherland, although they occasionally ventured on very

broken English for the instruction of the boy and girl which heaven had given them."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

31)

Sample:

By this time I had made several disquieting discoveries. The snow was falling faster than ever, the cold was increasing, a gale was blowing, and, under the circumstances, of course there was not a glimmer of light in the sky. My course was directly across the prairie, and in the event of my tracks being obliterated by the snow—as was almost certain to be the case—it was almost impossible for me to prevent myself from going astray.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9999

Generated Text:

The snow was falling faster than ever, the cold was increasing, a gale was blowing, and, under the circumstances, of course there was not a glimmer of light in the sky. My course was directly across the prairie, and in the event of my tracks being obliterated by the snow—as was almost certain to be the case—it was almost impossible for me to prevent myself from going astray. By this time I had made several disquieting discoveries.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """By this time I had made several disquieting discoveries. The snow was falling faster than ever, the cold was increasing,

a gale was blowing, and, under the circumstances, of course there was not a glimmer of light in the sky. My course was directly across

the prairie, and in the event of my tracks being obliterated by the snow—as was almost certain to be the case—it was almost impossible

for me to prevent myself from going astray."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.2782, METEOR Score: 0.9999

Generated Text:

The snow was falling faster than ever, the cold was increasing, a gale was blowing, and, under the circumstances, of course there was not a glimmer of light in the sky. My course was directly across the prairie, and in the event of my tracks being obliterated by the snow—as was almost certain to be the case—it was almost impossible for me to prevent myself from going astray. By this time I had made several disquieting discoveries.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install with: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """By this time I had made several disquieting discoveries. The snow was falling faster than ever, the cold was increasing,

a gale was blowing, and, under the circumstances, of course there was not a glimmer of light in the sky. My course was directly across

the prairie, and in the event of my tracks being obliterated by the snow—as was almost certain to be the case—it was almost impossible

for me to prevent myself from going astray."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

32)

Sample:

The following week the Atlantic sea-board was devastated by one of the fiercest storms that had been known for years. Reports of wrecks and disasters to shipping reached us for several days after, and Frank remarked one evening at supper that he believed his suspected pirate was one of the unfortunate vessels that had gone down with all on board. I smiled at his words, but when I learned that the beach was strewn with wreckage, and that a great deal of it had washed into the inlet, I thought it probable that he was right, so far as the fate of the strange ship was concerned.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.8477

Generated Text:

I smiled at his words, but when I learned that the beach was strewn with wreckage, and that a great deal of it had washed into the inlet, I thought it probable that he was right, so far as the fate of the strange ship was concerned. Reports of wrecks and disasters to shipping reached us for several days after, and Frank remarked one evening at supper that he believed his suspected pirate was one of the unfortunate vessels that had gone down with all on board. The following week the Atlantic sea-board was devastated by one of the fiercest storms that had been known for years.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The following week the Atlantic sea-board was devastated by one of the fiercest storms that had been known for years.

Reports of wrecks and disasters to shipping reached us for several days after, and Frank remarked one evening at supper that he believed

his suspected pirate was one of the unfortunate vessels that had gone down with all on board. I smiled at his words, but when I learned

that the beach was strewn with wreckage, and that a great deal of it had washed into the inlet, I thought it probable that he was right,

so far as the fate of the strange ship was concerned."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.8329, METEOR Score: 1.0000

Generated Text:

The following week the Atlantic sea-board was devastated by one of the fiercest storms that had been known for years. Reports of wrecks and disasters to shipping reached us for several days after, and Frank remarked one evening at supper that he believed his suspected pirate was one of the unfortunate vessels that had gone down with all on board. I smiled at his words, but when I learned that the beach was strewn with wreckage, and that a great deal of it had washed into the inlet, I thought it probable that he was right, so far as the fate of the strange ship was concerned.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # You can install pyswarm with: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The following week the Atlantic sea-board was devastated by one of the fiercest storms that had been known for years.

Reports of wrecks and disasters to shipping reached us for several days after, and Frank remarked one evening at supper that he believed

his suspected pirate was one of the unfortunate vessels that had gone down with all on board. I smiled at his words, but when I learned

that the beach was strewn with wreckage, and that a great deal of it had washed into the inlet, I thought it probable that he was right,

so far as the fate of the strange ship was concerned."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

33)

Sample:

On the following afternoon, when five o'clock arrived (in those days most of the country schools opened at eight and closed at five, with an hour at noon, and not more than two weeks vacation in summer. I have attended school on more than one Saturday, Fourth of July and Christmas), the school was all expectation. When Mr. Lathrop saw the bright eyes turned eagerly toward him, a thrill of pleasure stirred his heart, for he felt that his was the hand to sow good seed, or this was the soil where it could be made to spring up and bear fruit a hundred fold.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9830

Generated Text:

When Mr. Lathrop saw the bright eyes turned eagerly toward him, a thrill of pleasure stirred his heart, for he felt that his was the hand to sow good seed, or this was the soil where it could be made to spring up and bear fruit a hundred fold. On the following afternoon, when five o'clock arrived (in those days most of the country schools opened at eight and closed at five, with an hour at noon, and not more than two weeks vacation in summer. I have attended school on more than one Saturday, Fourth of July and Christmas), the school was all expectation.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """On the following afternoon, when five o'clock arrived (in those days most of the country schools opened at eight and

closed at five, with an hour at noon, and not more than two weeks vacation in summer. I have attended school on more than one Saturday,

Fourth of July and Christmas), the school was all expectation. When Mr. Lathrop saw the bright eyes turned eagerly toward him, a thrill

of pleasure stirred his heart, for he felt that his was the hand to sow good seed, or this was the soil where it could be made to spring

up and bear fruit a hundred fold."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.9663, METEOR Score: 0.9798

Generated Text:

I have attended school on more than one Saturday, Fourth of July and Christmas), the school was all expectation. When Mr. Lathrop saw the bright eyes turned eagerly toward him, a thrill of pleasure stirred his heart, for he felt that his was the hand to sow good seed, or this was the soil where it could be made to spring up and bear fruit a hundred fold. On the following afternoon, when five o'clock arrived (in those days most of the country schools opened at eight and closed at five, with an hour at noon, and not more than two weeks vacation in summer.

The code:

34)

Sample:

"No person can succeed in a business which he dislikes," remarked Mr. Hunter to Maggie who on this summer afternoon sat on the front porch, plying her deft needle, while the waning twilight lasted, with Bridget inside preparing the evening meal. "I think that is true, father," was her gentle reply.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

No person can succeed in a business which he dislikes," remarked Mr. Hunter to Maggie who on this summer afternoon sat on the front porch, plying her deft needle, while the waning twilight lasted, with Bridget inside preparing the evening meal. "I think that is true, father," was her gentle reply.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """No person can succeed in a business which he dislikes," remarked Mr. Hunter to Maggie who on this summer afternoon

sat on the front porch, plying her deft needle, while the waning twilight lasted, with Bridget inside preparing the evening meal.

"I think that is true, father," was her gentle reply."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5102, METEOR Score: 0.9992

Generated Text:

"I think that is true, father," was her gentle reply. No person can succeed in a business which he dislikes," remarked Mr. Hunter to Maggie who on this summer afternoon

sat on the front porch, plying her deft needle, while the waning twilight lasted, with Bridget inside preparing the evening meal.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install the library using: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """No person can succeed in a business which he dislikes," remarked Mr. Hunter to Maggie who on this summer afternoon

sat on the front porch, plying her deft needle, while the waning twilight lasted, with Bridget inside preparing the evening meal.

"I think that is true, father," was her gentle reply."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

35)

Sample:

The lad sat his horse like a skilled equestrian, and indeed it would be hard to find his superior in that respect throughout that broad stretch of sparsely settled country. Those who live on the American frontier are trained from their earliest youth in the management of quadrupeds, and often display a proficiency that cannot fail to excite admiration.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

The lad sat his horse like a skilled equestrian, and indeed it would be hard to find his superior in that respect throughout that broad stretch of sparsely settled country. Those who live on the American frontier are trained from their earliest youth in the management of quadrupeds, and often display a proficiency that cannot fail to excite admiration.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The lad sat his horse like a skilled equestrian, and indeed it would be hard to find his superior in that respect

throughout that broad stretch of sparsely settled country. Those who live on the American frontier are trained from their earliest

youth in the management of quadrupeds, and often display a proficiency that cannot fail to excite admiration."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.4577, METEOR Score: 1.0000

Generated Text:

The lad sat his horse like a skilled equestrian, and indeed it would be hard to find his superior in that respect throughout that broad stretch of sparsely settled country. Those who live on the American frontier are trained from their earliest youth in the management of quadrupeds, and often display a proficiency that cannot fail to excite admiration.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install the library using: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The lad sat his horse like a skilled equestrian, and indeed it would be hard to find his superior in that respect

throughout that broad stretch of sparsely settled country. Those who live on the American frontier are trained from their earliest

youth in the management of quadrupeds, and often display a proficiency that cannot fail to excite admiration."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

36)

Sample:

The animal was evidently puzzled at the sight before him. Fred dreaded a shot from the Indians above, and, as soon as he had his torch ready and had taken all his bearings, he drew the ashes over the spluttering flame. Save for the torch, all was again wrapped in impenetrable gloom.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9671

Generated Text:

Save for the torch, all was again wrapped in impenetrable gloom. Fred dreaded a shot from the Indians above, and, as soon as he had his torch ready and had taken all his bearings, he drew the ashes over the spluttering flame. The animal was evidently puzzled at the sight before him.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The animal was evidently puzzled at the sight before him. Fred dreaded a shot from the Indians above, and,

as soon as he had his torch ready and had taken all his bearings, he drew the ashes over the spluttering flame. Save for the torch,

all was again wrapped in impenetrable gloom."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.1780, METEOR Score: 0.9671

Generated Text:

Save for the torch, all was again wrapped in impenetrable gloom. Fred dreaded a shot from the Indians above, and, as soon as he had his torch ready and had taken all his bearings, he drew the ashes over the spluttering flame. The animal was evidently puzzled at the sight before him.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install this library using: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The animal was evidently puzzled at the sight before him. Fred dreaded a shot from the Indians above, and,

as soon as he had his torch ready and had taken all his bearings, he drew the ashes over the spluttering flame. Save for the torch,

all was again wrapped in impenetrable gloom."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO: Minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

37)

Sample:

Of course, this notice gave the whole scheme away, and some of the other town boys who pretended to make fun of us Scouts because we were trying to learn Scoutcraft and to use it right planned to cut us off and take the message away from us. There always are boys mean enough to bother and interfere, until they get to be Scouts themselves. Then they are ashamed.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

Of course, this notice gave the whole scheme away, and some of the other town boys who pretended to make fun of us Scouts because we were trying to learn Scoutcraft and to use it right planned to cut us off and take the message away from us. There always are boys mean enough to bother and interfere, until they get to be Scouts themselves. Then they are ashamed.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Of course, this notice gave the whole scheme away, and some of the other town boys who pretended to make fun

of us Scouts because we were trying to learn Scoutcraft and to use it right planned to cut us off and take the message away from us.

There always are boys mean enough to bother and interfere, until they get to be Scouts themselves. Then they are ashamed."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.0526, METEOR Score: 0.9290

Generated Text:

Then they are ashamed. There always are boys mean enough to bother and interfere, until they get to be Scouts themselves. Of course, this notice gave the whole scheme away, and some of the other town boys who pretended to make fun of us Scouts because we were trying to learn Scoutcraft and to use it right planned to cut us off and take the message away from us.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you have installed the pyswarm library with: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Of course, this notice gave the whole scheme away, and some of the other town boys who pretended to make fun

of us Scouts because we were trying to learn Scoutcraft and to use it right planned to cut us off and take the message away from us.

There always are boys mean enough to bother and interfere, until they get to be Scouts themselves. Then they are ashamed."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

38)

Sample:

Toward morning the stag got very cramped from keeping in one position so long. He moved his head slightly. In doing this his horns struck against the roof of the house. It made a terrible noise. The tiger thought that the stag was about to spring upon him and kill him. He made a leap for the door and ran out of it as fast as he could. He ran and ran until he was far, far away from the house with the roof of dried grass.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.8993

Generated Text:

He ran and ran until he was far, far away from the house with the roof of dried grass. The tiger thought that the stag was about to spring upon him and kill him. He made a leap for the door and ran out of it as fast as he could. He moved his head slightly. In doing this his horns struck against the roof of the house. Toward morning the stag got very cramped from keeping in one position so long. It made a terrible noise.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Toward morning the stag got very cramped from keeping in one position so long. He moved his head slightly.

In doing this his horns struck against the roof of the house. It made a terrible noise. The tiger thought that the stag was about to

spring upon him and kill him. He made a leap for the door and ran out of it as fast as he could. He ran and ran until he was far,

far away from the house with the roof of dried grass."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.4034, METEOR Score: 0.9615

Generated Text:

In doing this his horns struck against the roof of the house. It made a terrible noise. The tiger thought that the stag was about to spring upon him and kill him. He made a leap for the door and ran out of it as fast as he could. He moved his head slightly. He ran and ran until he was far, far away from the house with the roof of dried grass. Toward morning the stag got very cramped from keeping in one position so long.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you have installed the pyswarm library: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Toward morning the stag got very cramped from keeping in one position so long. He moved his head slightly.

In doing this his horns struck against the roof of the house. It made a terrible noise. The tiger thought that the stag was about to

spring upon him and kill him. He made a leap for the door and ran out of it as fast as he could. He ran and ran until he was far,

far away from the house with the roof of dried grass."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

39)

Sample:

At the coastguard station, midway between the Point and the village, they found the men on the alert, and two volunteered to go with Coomber and help man the boat. Then the four plodded silently along the slushy road, for talking was next to impossible in such a gale, and it needed all the strength and energy they could muster to fight the wind and rain.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

At the coastguard station, midway between the Point and the village, they found the men on the alert, and two volunteered to go with Coomber and help man the boat. Then the four plodded silently along the slushy road, for talking was next to impossible in such a gale, and it needed all the strength and energy they could muster to fight the wind and rain.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """At the coastguard station, midway between the Point and the village, they found the men on the alert,

and two volunteered to go with Coomber and help man the boat. Then the four plodded silently along the slushy road,

for talking was next to impossible in such a gale, and it needed all the strength and energy they could muster to fight the wind and rain."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.9226, METEOR Score: 1.0000

Generated Text:

At the coastguard station, midway between the Point and the village, they found the men on the alert, and two volunteered to go with Coomber and help man the boat. Then the four plodded silently along the slushy road, for talking was next to impossible in such a gale, and it needed all the strength and energy they could muster to fight the wind and rain.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install pyswarm with: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """At the coastguard station, midway between the Point and the village, they found the men on the alert,

and two volunteered to go with Coomber and help man the boat. Then the four plodded silently along the slushy road,

for talking was next to impossible in such a gale, and it needed all the strength and energy they could muster to fight the wind and rain."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

40)

Sample:

Mrs. Haydon was a widow, often ailing, and never strong enough to earn her own living by hard work, but through the kindness of her brother—himself not a wealthy man—a little business had been secured for her, enough to keep her in comfort, and he had urged that Kate, being young and strong, ought to get a situation.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 1.0000

Generated Text:

Mrs. Haydon was a widow, often ailing, and never strong enough to earn her own living by hard work, but through the kindness of her brother—himself not a wealthy man—a little business had been secured for her, enough to keep her in comfort, and he had urged that Kate, being young and strong, ought to get a situation.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Mrs. Haydon was a widow, often ailing, and never strong enough to earn her own living by hard work,

but through the kindness of her brother—himself not a wealthy man—a little business had been secured for her, enough

to keep her in comfort, and he had urged that Kate, being young and strong, ought to get a situation."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.2694, METEOR Score: 1.0000

Generated Text:

Mrs. Haydon was a widow, often ailing, and never strong enough to earn her own living by hard work, but through the kindness of her brother—himself not a wealthy man—a little business had been secured for her, enough to keep her in comfort, and he had urged that Kate, being young and strong, ought to get a situation.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # You need to install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Mrs. Haydon was a widow, often ailing, and never strong enough to earn her own living by hard work,

but through the kindness of her brother—himself not a wealthy man—a little business had been secured for her, enough

to keep her in comfort, and he had urged that Kate, being young and strong, ought to get a situation."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

41)

Sample:

The girls settled themselves to listen to Mary Louise's news, whatever it was. Elizabeth Wright closed her typewriter on which she had been copying some manuscript for a budding author; Irene Macfarlane stuck her needle in the pin-cushion hanging from her tidy work-basket and folded the lace collar. Only Josie went on with her work, testing her electric iron with a professional sizzle.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9840

Generated Text:

Only Josie went on with her work, testing her electric iron with a professional sizzle. The girls settled themselves to listen to Mary Louise's news, whatever it was. Elizabeth Wright closed her typewriter on which she had been copying some manuscript for a budding author; Irene Macfarlane stuck her needle in the pin-cushion hanging from her tidy work-basket and folded the lace collar.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The girls settled themselves to listen to Mary Louise's news, whatever it was.

Elizabeth Wright closed her typewriter on which she had been copying some manuscript for a budding author;

Irene Macfarlane stuck her needle in the pin-cushion hanging from her tidy work-basket and folded the lace collar.

Only Josie went on with her work, testing her electric iron with a professional sizzle."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.6707, METEOR Score: 0.9815

Generated Text:

The girls settled themselves to listen to Mary Louise's news, whatever it was. Only Josie went on with her work, testing her electric iron with a professional sizzle. Elizabeth Wright closed her typewriter on which she had been copying some manuscript for a budding author; Irene Macfarlane stuck her needle in the pin-cushion hanging from her tidy work-basket and folded the lace collar.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The girls settled themselves to listen to Mary Louise's news, whatever it was.

Elizabeth Wright closed her typewriter on which she had been copying some manuscript for a budding author;

Irene Macfarlane stuck her needle in the pin-cushion hanging from her tidy work-basket and folded the lace collar.

Only Josie went on with her work, testing her electric iron with a professional sizzle."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

42)

Sample:

Now when they had waited and waited a long, long time, and none had come back from the Dark Tower of Elfland, Childe Rowland, the youngest, the best beloved of Burd Helen's brothers, besought his mother to let him also go on the quest; for he was the bravest of them all, and neither death nor danger could dismay him. But at first his mother the Queen said: "Not so! You are the last of my children; if you are lost, all is lost indeed!"

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9840

Generated Text:

But at first his mother the Queen said: "Not so! You are the last of my children; if you are lost, all is lost indeed!" Now when they had waited and waited a long, long time, and none had come back from the Dark Tower of Elfland, Childe Rowland, the youngest, the best beloved of Burd Helen's brothers, besought his mother to let him also go on the quest; for he was the bravest of them all, and neither death nor danger could dismay him.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Now when they had waited and waited a long, long time, and none had come back from the Dark Tower of Elfland,

Childe Rowland, the youngest, the best beloved of Burd Helen's brothers, besought his mother to let him also go on the quest;

for he was the bravest of them all, and neither death nor danger could dismay him.

But at first his mother the Queen said: "Not so! You are the last of my children; if you are lost, all is lost indeed!" """

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.7608, METEOR Score: 0.9953

Generated Text:

But at first his mother the Queen said: "Not so! Now when they had waited and waited a long, long time, and none had come back from the Dark Tower of Elfland, Childe Rowland, the youngest, the best beloved of Burd Helen's brothers, besought his mother to let him also go on the quest; for he was the bravest of them all, and neither death nor danger could dismay him. You are the last of my children; if you are lost, all is lost indeed!"

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """Now when they had waited and waited a long, long time, and none had come back from the Dark Tower of Elfland,

Childe Rowland, the youngest, the best beloved of Burd Helen's brothers, besought his mother to let him also go on the quest;

for he was the bravest of them all, and neither death nor danger could dismay him.

But at first his mother the Queen said: "Not so! You are the last of my children; if you are lost, all is lost indeed!" """

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

43)

Sample:

The Great Spirit thought, "By and by I will make men, but first I will make a home for them. It shall be very bright and beautiful. There shall be mountains and prairies and forests, and about it all shall be the blue waters of the sea."

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9983

Generated Text:

It shall be very bright and beautiful. The Great Spirit thought, "By and by I will make men, but first I will make a home for them. There shall be mountains and prairies and forests, and about it all shall be the blue waters of the sea."

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The Great Spirit thought, "By and by I will make men, but first I will make a home for them.

It shall be very bright and beautiful. There shall be mountains and prairies and forests, and about it all shall be the blue waters of the sea." """

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.4374, METEOR Score: 0.9670

Generated Text:

The Great Spirit thought, "By and by I will make men, but first I will make a home for them. There shall be mountains and prairies and forests, and about it all shall be the blue waters of the sea." It shall be very bright and beautiful.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure you install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """The Great Spirit thought, "By and by I will make men, but first I will make a home for them.

It shall be very bright and beautiful. There shall be mountains and prairies and forests, and about it all shall be the blue waters of the sea." """

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

44)

Sample:

One day the Great Spirit asked all the animals that he had made to come to his lodge. Those that could fly came first: the robin, the bluebird, the owl, the butterfly, the wasp, and the firefly. Behind them came the chicken, fluttering its wings and trying hard to keep up. Then came the deer, the squirrel, the serpent, the cat, and the rabbit. Last of all came the bear, the beaver, and the hedgehog. Every one traveled as swiftly as he could, for each wished to hear the words of the Great Spirit.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.7753

Generated Text:

Every one traveled as swiftly as he could, for each wished to hear the words of the Great Spirit. Behind them came the chicken, fluttering its wings and trying hard to keep up. One day the Great Spirit asked all the animals that he had made to come to his lodge. Then came the deer, the squirrel, the serpent, the cat, and the rabbit. Those that could fly came first: the robin, the bluebird, the owl, the butterfly, the wasp, and the firefly. Last of all came the bear, the beaver, and the hedgehog.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """One day the Great Spirit asked all the animals that he had made to come to his lodge.

Those that could fly came first: the robin, the bluebird, the owl, the butterfly, the wasp, and the firefly.

Behind them came the chicken, fluttering its wings and trying hard to keep up. Then came the deer, the squirrel, the serpent, the cat, and the rabbit.

Last of all came the bear, the beaver, and the hedgehog. Every one traveled as swiftly as he could, for each wished to hear the words of the Great Spirit."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5187, METEOR Score: 0.7658

Generated Text:

Every one traveled as swiftly as he could, for each wished to hear the words of the Great Spirit. One day the Great Spirit asked all the animals that he had made to come to his lodge. Then came the deer, the squirrel, the serpent, the cat, and the rabbit. Last of all came the bear, the beaver, and the hedgehog. Behind them came the chicken, fluttering its wings and trying hard to keep up. Those that could fly came first: the robin, the bluebird, the owl, the butterfly, the wasp, and the firefly.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Install pyswarm via: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# New original passage

original\_text = """One day the Great Spirit asked all the animals that he had made to come to his lodge.

Those that could fly came first: the robin, the bluebird, the owl, the butterfly, the wasp, and the firefly.

Behind them came the chicken, fluttering its wings and trying hard to keep up. Then came the deer, the squirrel, the serpent, the cat, and the rabbit.

Last of all came the bear, the beaver, and the hedgehog. Every one traveled as swiftly as he could, for each wished to hear the words of the Great Spirit."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    return " ".join(selected\_sentences)

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

45)

Sample:

Then the birds tried their wings. They flew high, high up above the earth, but one by one they had to come back to their homes. It was soon seen which could fly highest, for when all the others had come back, there was the eagle rising higher and higher.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9803

Generated Text:

It was soon seen which could fly highest, for when all the others had come back, there was the eagle rising higher and higher. Then the birds tried their wings. They flew high, high up above the earth, but one by one they had to come back to their homes.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Then the birds tried their wings. They flew high, high up above the earth, but one by one they had to come back to their homes.

It was soon seen which could fly highest, for when all the others had come back, there was the eagle rising higher and higher."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Temperature: 1.000, Length: 100, METEOR Score: 0.9803

Generated Text:

It was soon seen which could fly highest, for when all the others had come back, there was the eagle rising higher and higher. Then the birds tried their wings. They flew high, high up above the earth, but one by one they had to come back to their homes.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Then the birds tried their wings. They flew high, high up above the earth, but one by one they had to come back to their homes.

It was soon seen which could fly highest, for when all the others had come back, there was the eagle rising higher and higher."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

46)

Sample:

In a country that is far away there once lived a young man called Tithonus. He was strong and beautiful. Light of heart and light of foot, he hunted the deer or danced and sang the livelong day. Every one who saw him loved him, but the one that loved him most was a goddess named Aurora.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9953

Generated Text:

He was strong and beautiful. Light of heart and light of foot, he hunted the deer or danced and sang the livelong day. Every one who saw him loved him, but the one that loved him most was a goddess named Aurora. In a country that is far away there once lived a young man called Tithonus.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """In a country that is far away there once lived a young man called Tithonus. He was strong and beautiful.

Light of heart and light of foot, he hunted the deer or danced and sang the livelong day. Every one who saw him loved him, but

the one that loved him most was a goddess named Aurora."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.3742, METEOR Score: 0.9196

Generated Text:

Every one who saw him loved him, but the one that loved him most was a goddess named Aurora. Light of heart and light of foot, he hunted the deer or danced and sang the livelong day. In a country that is far away there once lived a young man called Tithonus. He was strong and beautiful.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Ensure pyswarm is installed: pip install pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """In a country that is far away there once lived a young man called Tithonus. He was strong and beautiful.

Light of heart and light of foot, he hunted the deer or danced and sang the livelong day. Every one who saw him loved him, but

the one that loved him most was a goddess named Aurora."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

47)

Sample:

Everyday Tithonus grew older and smaller. "I am no longer happy in your father's home," he said, "with your brothers who are as beautiful and as strong as I was when I first saw you. Let me go back to my own country. Let me be a bird or an insect and live in the fields where we first roamed together. Let me go, dearest goddess."

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9929

Generated Text:

Let me go back to my own country. Let me be a bird or an insect and live in the fields where we first roamed together. 'I am no longer happy in your father's home,' he said, 'with your brothers who are as beautiful and as strong as I was when I first saw you. Everyday Tithonus grew older and smaller. Let me go, dearest goddess.'

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Everyday Tithonus grew older and smaller. 'I am no longer happy in your father's home,' he said,

'with your brothers who are as beautiful and as strong as I was when I first saw you. Let me go back to my own country.

Let me be a bird or an insect and live in the fields where we first roamed together. Let me go, dearest goddess.'"""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5002, METEOR Score: 0.9962

Generated Text:

'I am no longer happy in your father's home,' he said, 'with your brothers who are as beautiful and as strong as I was when I first saw you. Let me be a bird or an insect and live in the fields where we first roamed together. Let me go, dearest goddess.' Everyday Tithonus grew older and smaller. Let me go back to my own country.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Import PSO from pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Everyday Tithonus grew older and smaller. 'I am no longer happy in your father's home,' he said,

'with your brothers who are as beautiful and as strong as I was when I first saw you. Let me go back to my own country.

Let me be a bird or an insect and live in the fields where we first roamed together. Let me go, dearest goddess.'"""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

48)

Sample:

Winter was coming, and the birds had flown far to the south, where the air was warm and they could find berries to eat. One little bird had broken its wing and could not fly with the others. It was alone in the cold world of frost and snow. The forest looked warm, and it made its way to the trees as well as it could, to ask for help.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.8388

Generated Text:

One little bird had broken its wing and could not fly with the others. The forest looked warm, and it made its way to the trees as well as it could, to ask for help. It was alone in the cold world of frost and snow. Winter was coming, and the birds had flown far to the south, where the air was warm and they could find berries to eat.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Winter was coming, and the birds had flown far to the south, where the air was warm and they could find berries to eat.

One little bird had broken its wing and could not fly with the others. It was alone in the cold world of frost and snow.

The forest looked warm, and it made its way to the trees as well as it could, to ask for help."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.3796, METEOR Score: 0.9744

Generated Text:

It was alone in the cold world of frost and snow. Winter was coming, and the birds had flown far to the south, where the air was warm and they could find berries to eat. One little bird had broken its wing and could not fly with the others. The forest looked warm, and it made its way to the trees as well as it could, to ask for help.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Import PSO from pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """Winter was coming, and the birds had flown far to the south, where the air was warm and they could find berries to eat.

One little bird had broken its wing and could not fly with the others. It was alone in the cold world of frost and snow.

The forest looked warm, and it made its way to the trees as well as it could, to ask for help."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

49)

Sample:

In the morning all those shining green leaves lay on the ground, for a cold north wind had come in the night, and every leaf that it touched fell from the tree. "May I touch every leaf in the forest?" asked the wind in its frolic. "No," said the frost king. "The trees that have been kind to the little bird with the broken wing may keep their leaves." This is why the leaves of the spruce, the pine, and the juniper are always green.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9967

Generated Text:

In the morning all those shining green leaves lay on the ground, for a cold north wind had come in the night, and every leaf that it touched fell from the tree. "No," said the frost king. "May I touch every leaf in the forest?" "The trees that have been kind to the little bird with the broken wing may keep their leaves." asked the wind in its frolic. This is why the leaves of the spruce, the pine, and the juniper are always green.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """In the morning all those shining green leaves lay on the ground, for a cold north wind had come in the night,

and every leaf that it touched fell from the tree. "May I touch every leaf in the forest?" asked the wind in its frolic.

"No," said the frost king. "The trees that have been kind to the little bird with the broken wing may keep their leaves."

This is why the leaves of the spruce, the pine, and the juniper are always green."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.1985, METEOR Score: 0.9258

Generated Text:

asked the wind in its frolic. "No," said the frost king. This is why the leaves of the spruce, the pine, and the juniper are always green. "May I touch every leaf in the forest?" In the morning all those shining green leaves lay on the ground, for a cold north wind had come in the night, and every leaf that it touched fell from the tree. "The trees that have been kind to the little bird with the broken wing may keep their leaves."

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Import PSO from pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """In the morning all those shining green leaves lay on the ground, for a cold north wind had come in the night,

and every leaf that it touched fell from the tree. "May I touch every leaf in the forest?" asked the wind in its frolic.

"No," said the frost king. "The trees that have been kind to the little bird with the broken wing may keep their leaves."

This is why the leaves of the spruce, the pine, and the juniper are always green."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)

50)

Sample:

The gentleman with the kind eyes looked at Elizabeth with an interested expression as she came into the room. Her slender little figure in its black velvet dress, her delicate little face with its large soft sad eyes, the gentle gravity of her manner made her seem quite unlike other children.

Manually:

Temperature: 1.0, Length: 100, METEOR Score: 0.9897

Generated Text:

Her slender little figure in its black velvet dress, her delicate little face with its large soft sad eyes, the gentle gravity of her manner made her seem quite unlike other children. The gentleman with the kind eyes looked at Elizabeth with an interested expression as she came into the room.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """The gentleman with the kind eyes looked at Elizabeth with an interested expression as she came into the room.

Her slender little figure in its black velvet dress, her delicate little face with its large soft sad eyes, the gentle gravity of her manner

made her seem quite unlike other children."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    # Tokenize the original text into sentences

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Fixed parameters

chosen\_temp = 1.0  # Temperature set to 1 for creativity

chosen\_length = 100

# Generate text and calculate METEOR score

generated\_text = generate\_response(original\_text, chosen\_temp, chosen\_length)

generated\_tokens = generated\_text.split()

# Compute METEOR score

score = meteor\_score([original\_text.split()], generated\_tokens)

# Print results

print(f"Temperature: {chosen\_temp}, Length: {chosen\_length}, METEOR Score: {score:.4f}")

print("\nGenerated Text:\n")

print(generated\_text)  # This prints the generated text

Optimized:

Optimized Temperature: 0.5675, METEOR Score: 0.9897

Generated Text:

Her slender little figure in its black velvet dress, her delicate little face with its large soft sad eyes, the gentle gravity of her manner made her seem quite unlike other children. The gentleman with the kind eyes looked at Elizabeth with an interested expression as she came into the room.

The code:

import nltk

from nltk.translate.meteor\_score import meteor\_score

import random

from pyswarm import pso  # Import PSO from pyswarm

# Ensure necessary resources are downloaded

nltk.download('wordnet', download\_dir='./nltk\_data')

nltk.download('punkt', download\_dir='./nltk\_data')

# Add the download directory to NLTK's data path

nltk.data.path.append('./nltk\_data')

# Original passage

original\_text = """The gentleman with the kind eyes looked at Elizabeth with an interested expression as she came into the room.

Her slender little figure in its black velvet dress, her delicate little face with its large soft sad eyes, the gentle gravity of her manner

made her seem quite unlike other children."""

# Tokenize the original text into sentences

sentences = nltk.sent\_tokenize(original\_text)

# Function to generate text with temperature and response length

def generate\_response(original\_text, temperature=1.0, response\_length=100):

    sentences = nltk.sent\_tokenize(original\_text)

    # Apply temperature adjustment (higher temperature results in more randomness)

    if temperature < 1.0:

        sentences = sorted(sentences, key=lambda \_: random.random() \*\* (1 / temperature))

    # Generate the response by randomly sampling sentences and ensuring response length

    num\_sentences = max(1, response\_length // 10)  # Ensure at least one sentence is selected

    selected\_sentences = random.sample(sentences, min(num\_sentences, len(sentences)))

    # Join the selected sentences into a coherent passage

    generated\_text = " ".join(selected\_sentences)

    return generated\_text

# Objective function for PSO to minimize METEOR score

def objective\_function(temp):

    temperature = temp[0]  # Extract temperature value from PSO

    generated\_text = generate\_response(original\_text, temperature=temperature, response\_length=100)

    generated\_tokens = generated\_text.split()

    # Compute METEOR score (lower is better)

    score = meteor\_score([original\_text.split()], generated\_tokens)

    return score

# Set bounds for temperature optimization (between 0 and 1)

lb = [0.01]  # Avoid zero to ensure there is some randomness

ub = [1.0]   # Full randomness

# Run PSO to optimize temperature for the lowest METEOR score

best\_temp, \_ = pso(objective\_function, lb, ub)

# Optimized temperature found by PSO

optimized\_temp = best\_temp[0]

# Generate text using the optimized temperature

final\_text = generate\_response(original\_text, temperature=optimized\_temp, response\_length=100)

final\_tokens = final\_text.split()

# Compute final METEOR score

final\_score = meteor\_score([original\_text.split()], final\_tokens)

# Print results

print(f"Optimized Temperature: {optimized\_temp:.4f}, METEOR Score: {final\_score:.4f}")

print("\nGenerated Text:\n")

print(final\_text)