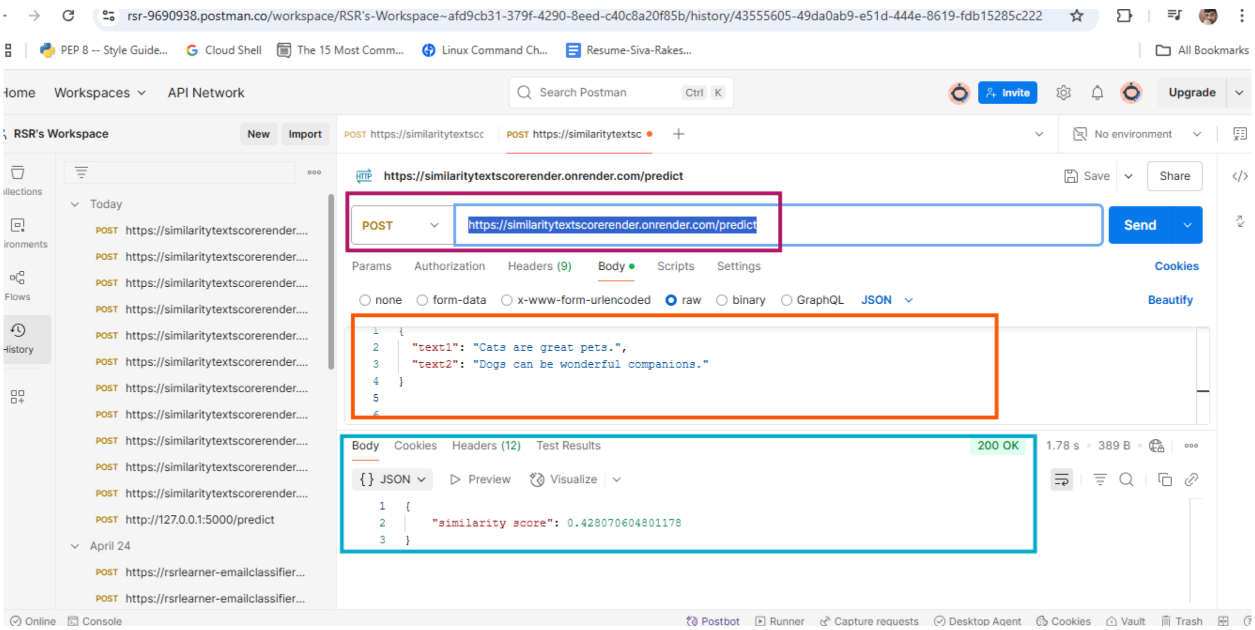
Semantic Text Similarity – Technical Report

API Link : https://similaritytextscorerender.onrender.com/predict

Name: Relangi Siva Rakesh
Contact: [Phone Number → +91 9441965520 | Email Address → Sivarakesh993@gmail.com]



https://similaritytextscorerender.onrender.com/predict

POST https://similaritytextscorerender.onrender.com/predict

Body: raw, JSON

```
{
  "text1": "Cats are great pets.",
  "text2": "Dogs can be wonderful companions."
}
```

200 OK   1.78 s   389 B

```
{
    "similarity score": 0.428070604801178
}
```

Part A – Core Approach: Text Similarity with Sentence-BERT

1. Introduction

This report outlines the methodology and implementation of a semantic text similarity API that evaluates how similar two pieces of text are, based on meaning rather than exact wording. The approach leverages **Sentence-BERT (SBERT)**, a transformer-based model designed to generate sentence embeddings suitable for similarity tasks.

2. Model Selection

The API uses the **`paraphrase-MiniLM-L3-v2`** model from `SentenceTransformer` library. Chosen for its:

- Compact size (~14MB), ideal for deployment on limited-resource platforms.

- Balance between **speed** and **semantic accuracy**.
- It is compatible with free-tier services due to its lightweight footprint

3. Workflow Overview

Step 1: Text Encoding

Both input texts are encoded using SBERT:

```
model.encode([text1, text2])
```

This produces two dense vector embeddings representing the semantic content of the texts.

Step 2: Similarity Calculation

Cosine similarity is used to measure how close the two embeddings are in high-dimensional space:

```
util.cos_sim(embedding1, embedding2)
```

- Result: A float value between **0 (no similarity)** and **1 (identical meaning)**.

Step 3: Output

The API returns a JSON response:

```
{
    "similarity score": 0.8832
}
```

Note: The model internally handles preprocessing like lowercasing, tokenization, and padding.

Part B – API Deployment

1. API Design

- Framework: **Flask**
- Endpoint: `POST /predict`
- Input: JSON body with two fields – `text1`, `text2`
- Output: JSON response with a similarity score

2. Example Request (Postman or cURL)

POST `http://127.0.0.1:5000/predict` or https://similaritytextscorerender.onrender.com/predict

Body (raw JSON):

```
{
    "text1": "The quick brown fox jumps over the lazy dog.",
    "text2": "A fast fox leaps over a lazy dog."
}
```

Response:

```
{
    "similarity score": 0.8832
}
```

3. Deployment Platforms Tested

Platform	Status	Issue
AWS EC2 (Free)	❌ Failed	Insufficient disk space for Hugging Face model downloads
GCP (Free)	❌ Failed	Disk quota exceeded; storage limits
Heroku (Free)	❌ Failed	Dyno memory/storage too low for model load
PythonAnywhere	❌ Failed	Disk usage limitations prevent loading the model

Note: All failures are due to **storage constraints** of free-tier plans.

Alternative Deployment: Render (Success ✅)

Why Render?

- Supports containerized deployments.
- Generous free-tier limits compared to other platforms.
- Handles dependencies and background processes smoothly.

Configuration

- `render.yaml` is used for deployment settings.
- **Gunicorn** is used as a production-grade WSGI server.
- The model is dynamically downloaded from Hugging Face at first request.

Conclusion

This project successfully demonstrates how **Sentence-BERT** can be used to compute text similarity in an unsupervised manner. Despite challenges with cloud deployment on common free-tier platforms, the system was successfully hosted on **Render**, thanks to its better resource allocations. The solution provides:

- Fast, semantic similarity results.
- Lightweight architecture suitable for scalable deployment.
- No need for labeled training data.