**all-MiniLM-L6-v2** is a pre-trained language model specifically designed for generating high-quality sentence embeddings. This means it can convert sentences and paragraphs into numerical representations (vectors) that capture their semantic meaning.

In essence, all-MiniLM-L6-v2 is a powerful tool for understanding and processing text data at the sentence level.

**Steps:**

* **Prepare the Data**:
	+ Extract the professor's posts (parent IDs and messages).
	+ Extract student replies corresponding to each parent post (message where parent = prof\_id).
* **Embed the Messages**:
	+ Use a pre-trained sentence embedding model (e.g., SentenceTransformer).
	+ Generate embeddings for both professor's posts and student replies.
* **Compute Similarity**:
	+ For each student reply, calculate the cosine similarity between the reply's embedding and the professor's post's embedding.
* **Store and Output Scores**:
	+ Save the relevance score for each student reply in the database or output the results as a report.

import sqlite3

import pandas as pd

from sentence\_transformers import SentenceTransformer

from sklearn.metrics.pairwise import cosine\_similarity

*# Connect to SQLite database*

conn = sqlite3.connect('Moodle\_Act.db')

*# Query professor posts and student replies*

prof\_query = "SELECT id AS prof\_id, message AS prof\_message FROM discussions\_m WHERE userid = 34004"  *# Professor's posts*

reply\_query = "SELECT id, parent, userid, userfullname, message AS student\_message FROM discussions\_m WHERE parent IN (SELECT id FROM discussions\_m WHERE userid = 34004)"  *# Student replies*

prof\_posts = pd.read\_sql\_query(prof\_query, conn)

student\_replies = pd.read\_sql\_query(reply\_query, conn)

conn.close()

*# Load pre-trained model*

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

*# Handle missing or null messages*

prof\_posts['prof\_message'] = prof\_posts['prof\_message'].fillna('')

prof\_posts = prof\_posts[prof\_posts['prof\_message'] != '']  *# Remove empty strings if necessary*

*# Generate embeddings for professor posts*

prof\_posts['embedding'] = prof\_posts['prof\_message'].apply(**lambda** x: model.encode(x) if isinstance(x, str) and x.strip() else None)

*# Filter out replies with invalid parent IDs*

student\_replies = student\_replies[student\_replies['parent'].isin(prof\_posts['prof\_id'])]

*# Function to calculate similarity*

**def** calculate\_similarity(reply, prof\_posts):

    prof\_id = reply['parent']

    prof\_embeddings = prof\_posts.loc[prof\_posts['prof\_id'] == prof\_id, 'embedding']

    if prof\_embeddings.empty:  *# If no matching professor post is found*

        return None

    prof\_embedding = prof\_embeddings.values[0]

    reply\_embedding = model.encode(reply['student\_message'])

    similarity = cosine\_similarity([prof\_embedding], [reply\_embedding])[0][0]

    return similarity

*# Compute similarity for each student reply*

student\_replies['topic\_relevance\_score'] = student\_replies.apply(

    **lambda** x: calculate\_similarity(x, prof\_posts) if x['student\_message'] else None, axis=1

)

*# Merge professor messages with student replies for context*

merged\_df = pd.merge(

    student\_replies,

    prof\_posts[['prof\_id', 'prof\_message']],

    left\_on='parent',

    right\_on='prof\_id',

    how='left'

)

*# Save results to a CSV file*

merged\_df[['userid', 'userfullname', 'student\_message', 'prof\_message', 'topic\_relevance\_score']].to\_csv(

    'topic\_relevance\_scores\_with\_messages.csv',

    index=False

)

*# Optional: Store back in SQLite database*

conn = sqlite3.connect('Moodle\_Act.db')

merged\_df[['id', 'topic\_relevance\_score']].to\_sql('relevance\_scores', conn, if\_exists='replace', index=False)

conn.close()