NOTE to self : Use COLAB

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import torch
from sentence_transformers import SentenceTransformer
import torch
print(torch.cuda.is_available())
print(torch.cuda.get_device_name(0))
```

```
True
Tesla T4
```

📘 What This Dataset Is and What It Contains

- `maartengr/arxiv_nlp` is a curated subset of arXiv papers focused **only on the field of NLP** (the arXiv category **cs.CL – Computation and Language**).
- It contains **tens of thousands of papers** published between **1991 and 2024**.
- Each record represents **one arXiv paper** and includes:
  - **Title** – the paper's name
  - **Abstracts** – the full abstract text
  - **Years** – publication year
  - **Categories** – always `"Computation and Language"` (because this dataset is filtered to cs.CL only)
- The dataset is already cleaned and structured, so you don't download PDFs or raw text — only metadata and text fields ready for NLP tasks.

```
# Load data from huggingface
from datasets import load_dataset
dataset = load_dataset("maartengr/arxiv_nlp")["train"]
# Extract specific metadata
abstracts = dataset["Abstracts"]
df = pd.DataFrame(dataset)
years = dataset["Years"]
categories = dataset["Categories"]
titles = dataset["Titles"]
print(df.shape)
print(df.info())
df.head()


```

```
(44949, 4)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44949 entries, 0 to 44948
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Titles      44949 non-null  object
 1   Abstracts   44949 non-null  object
 2   Years       44949 non-null  int64 
 3   Categories  44949 non-null  object
dtypes: int64(1), object(3)
memory usage: 1.4+ MB
None
```

	Titles	Abstracts	Years	Categories
0	Introduction to Arabic Speech Recognition Usin...	In this paper Arabic was investigated from t...	2007	Computation and Language
1	Arabic Speech Recognition System using CMU-Sph...	In this paper we present the creation of an ...	2007	Computation and Language
2	On the Development of Text Input Method - Less...	Intelligent Input Methods (IM) are essential...	2007	Computation and Language
3	Network statistics on early English Syntax: St...	This paper includes a reflection on the role...	2007	Computation and Language
4	Segmentation and Context of Literary and Music...	We test a segmentation algorithm, based on t...	2007	Computation and Language

Next steps: Generate code with df | New interactive sheet

🔍 What this code does

1. Loads the **ArXiv NLP dataset** from HuggingFace.

2. Loads a **very small SentenceTransformer model** (`all-MiniLM-L6-v1`) to create text embeddings.
3. Converts each abstract into a **384-dimensional embedding vector**.
4. Applies **PCA (Principal Component Analysis)** to reduce the embeddings from 384 dimensions down to **5 dimensions**.
    - PCA is used instead of UMAP because it is **much faster on CPU**.
5. Prints the original embedding shape and the reduced shape.

What the output means

- `Original shape: (10000, 384)`
  → We created embeddings for 10,000 documents, each with 384 features.
- `Reduced shape: (10000, 5)`
  → PCA compressed each embedding to 5 numbers while keeping the main structure.

These reduced embeddings can later be used for **clustering**, **visualization**, or **topic modeling** (like BERTopic), but they are much lighter and faster to process.

What does it mean to embed the articles?

Embedding an article means converting its text into a vector of numbers that captures its meaning.
Similar articles get similar vectors; different topics get distant vectors.
This lets us compare, cluster, or visualize the articles mathematically.

How does the model create these embeddings?

1. The model was pretrained on millions of texts, so it already understands language patterns.
2. It splits the article into tokens (words or subwords).
3. A transformer model processes how each word relates to the others using attention.
4. It combines everything into one final vector (e.g., 384 numbers) that represents the meaning of the whole article.

In short: the model gives each article a **numerical "meaning fingerprint"** based on what it learned during pretraining.

```python
import torch
import numpy as np
from sentence_transformers import SentenceTransformer
from umap import UMAP

# Make list of documents
abstract_list = list(dataset["Abstracts"])

# Device
device = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", device)

# Embedding model on device
model = SentenceTransformer("all-MiniLM-L6-v1", device=device)

# Create embeddings
emb = model.encode(
    abstract_list,
    batch_size=128,
    show_progress_bar=True,
    convert_to_numpy=True
)

# UMAP (same style as your textbook)
umap_model = UMAP(
    n_neighbors=15,
    n_components=5,
    min_dist=0.0,
    metric="cosine"
)

# Reduce embeddings
reduced_embeddings = umap_model.fit_transform(emb)

print("Original shape:", emb.shape)
print("Reduced shape:", reduced_embeddings.shape)

```

```
Using device: cuda
Batches: 100%                                                352/352 [01:16<00:00,  7.79it/s]
Original shape: (44949, 384)
Reduced shape: (44949, 5)
```

```python
print("embeddings of frist article in the dataset: ")
print(emb[0])
```

```
print("-"*100)
print("reduced_embeddings of frist article in the dataset: ")
print(reduced_embeddings[0])
```

```
-3.83223779e-02  1.43729309e-02 -1.24183102e-02 -5.89965284e-02
-5.18174618e-02  1.92359909e-02  1.15772940e-01 -2.91441642e-02
-2.92759147e-02  6.71013743e-02  2.57339999e-02  6.67332381e-04
-4.03684936e-02 -2.10710913e-02  3.31643932e-02  1.75768547e-02
-1.74630340e-02  3.96120362e-02  4.83366400e-02 -5.47545590e-02
-3.09314369e-03  4.04474624e-02  3.94865051e-02 -2.59100832e-03
-5.68059832e-03 -2.27152146e-02  5.59130348e-02  3.34894545e-02
 6.08115569e-02 -3.04217171e-02  4.85400148e-02  6.94097653e-02
 6.16980344e-02  3.23564857e-02 -3.82243842e-03  5.47760837e-02
 6.33379519e-02  8.56095329e-02 -1.23063900e-01  6.22815602e-02
 3.72754014e-03 -5.10304458e-02 -4.42683324e-02  1.59919113e-02
 6.26467634e-03  6.31978065e-02  3.38399690e-03  4.52167876e-02
 3.29400897e-02  1.04321148e-02  5.61873652e-02 -2.66926941e-02
-2.37167589e-02  6.35937834e-03  5.10728266e-03  8.72034058e-02
-7.29048848e-02  7.37879723e-02  5.52615663e-03  2.11064033e-33
-2.37486046e-02  4.80232686e-02  2.14588270e-02  7.51758367e-02
-7.57525535e-03 -5.96167929e-02  8.86392966e-02  7.49479458e-02
-2.44155060e-03 -2.54827570e-02 -1.61467046e-02 -6.24164764e-04
 6.48203641e-02 -6.32777214e-02  8.88504907e-02 -3.89383473e-02
-8.17993656e-03  4.33328785e-02  2.38112286e-02  1.17860429e-01
-1.97774805e-02  1.27070127e-02 -9.99852940e-02 -5.07258810e-02
 1.66946091e-02 -8.86317168e-04 -5.58230691e-02  2.54349373e-02
-1.70671772e-02  7.77923018e-02 -3.14253680e-02 -7.62655167e-03
-1.67943567e-01  5.79814836e-02 -4.82146405e-02 -4.78509702e-02
-7.34072551e-02 -2.08757967e-02 -4.65500634e-03  6.60007149e-02
 8.54691118e-02  4.90958169e-02 -5.73803969e-02 -1.02964386e-01
 4.89517972e-02  1.45720772e-03 -9.99710783e-02  6.83254600e-02
-8.31274167e-02 -2.55559813e-02  3.73049192e-02  3.15551355e-04
-8.06467608e-03  1.11036701e-03  2.37282738e-02 -2.98973639e-03
-4.25360836e-02  1.58171542e-02 -2.18579620e-02 -2.92654485e-02
-4.21719998e-02 -7.27164820e-02  6.21571578e-02 -4.97371927e-02
-3.52145941e-03  1.48062119e-02 -5.70303462e-02 -2.23721638e-02
-2.06010565e-02 -5.53027466e-02  2.60043088e-02 -9.50312149e-03
-5.32760695e-02  3.53901237e-02 -3.99958044e-02  6.39839321e-02
-1.00381896e-01  1.89229734e-02 -8.99495333e-02 -3.95809896e-02
 5.75916376e-03  1.51210483e-02  5.32058366e-02  7.83055276e-02
 9.91253480e-02  7.20499754e-02  4.67104428e-02 -5.45558892e-02
 2.00802181e-02  2.98801176e-02 -3.78906578e-02  5.25860973e-02
 7.25908279e-02  8.29531327e-02 -5.21017946e-02 -4.02759701e-20
-7.99557939e-02  3.96862403e-02  8.67416933e-02 -9.50825810e-02
-3.88660878e-02  3.00695114e-02  9.98179428e-03 -6.20949417e-02
 2.47521568e-02 -1.03646189e-01  7.51466379e-02 -7.60885626e-02
-5.76395951e-02  5.03646769e-02 -1.18760966e-01  5.35026975e-02
 8.26581940e-02 -2.13911440e-02  1.35946730e-02 -9.71064046e-02
 7.10243881e-02 -1.11592896e-02  8.77325609e-03  7.53341094e-02
-9.02557299e-02  3.58231179e-02 -2.76436321e-02  5.79884201e-02
-5.03160479e-03 -9.30505767e-02  1.62647839e-03  2.30145808e-02
-6.45225123e-02 -5.65111812e-04  4.38102521e-02  2.24746317e-02
 1.31918117e-04 -2.70254351e-02  2.26985794e-02 -1.89936142e-02
 5.30378409e-02 -5.11558168e-02 -1.08483189e-03 -2.91613005e-02
 1.78610301e-03  8.46299902e-02 -2.62068156e-02 -9.01427492e-02
-7.77656063e-02  3.16463374e-02  5.29958382e-02 -1.88826416e-02
 4.60220985e-02  5.05105443e-02  2.31922399e-02 -1.28954835e-02
 3.66763286e-02 -3.99592668e-02 -3.89709859e-03  5.56191318e-02
 9.07307211e-03 -1.46849835e-02  1.95369264e-03 -6.10346086e-02]
----------------------------------------------------------------------------------------------------
reduced_embeddings of frist article in the dataset:
[8.576501  9.607148  3.8321784 2.0628989 6.2317533]
```

```
from hdbscan import HDBSCAN

# Instantiate HDBSCAN model
hdbscan_model = HDBSCAN(min_cluster_size=15, metric='euclidean')

# Fit on the reduced UMAP embeddings
hdbscan_model.fit(reduced_embeddings)

# Cluster labels
labels = hdbscan_model.labels_
print(labels)
```

```
[ -1  -1  -1 ...  -1 181  -1]
```

```
import seaborn as sns
import pandas as pd
import numpy as np
from umap import UMAP

# Reduce original embeddings to 2 dimensions for visualization
umap_2d = UMAP(
    n_neighbors=15,
    n_components=2,
    min_dist=0.0,
    metric='cosine'
).fit_transform(emb)

```

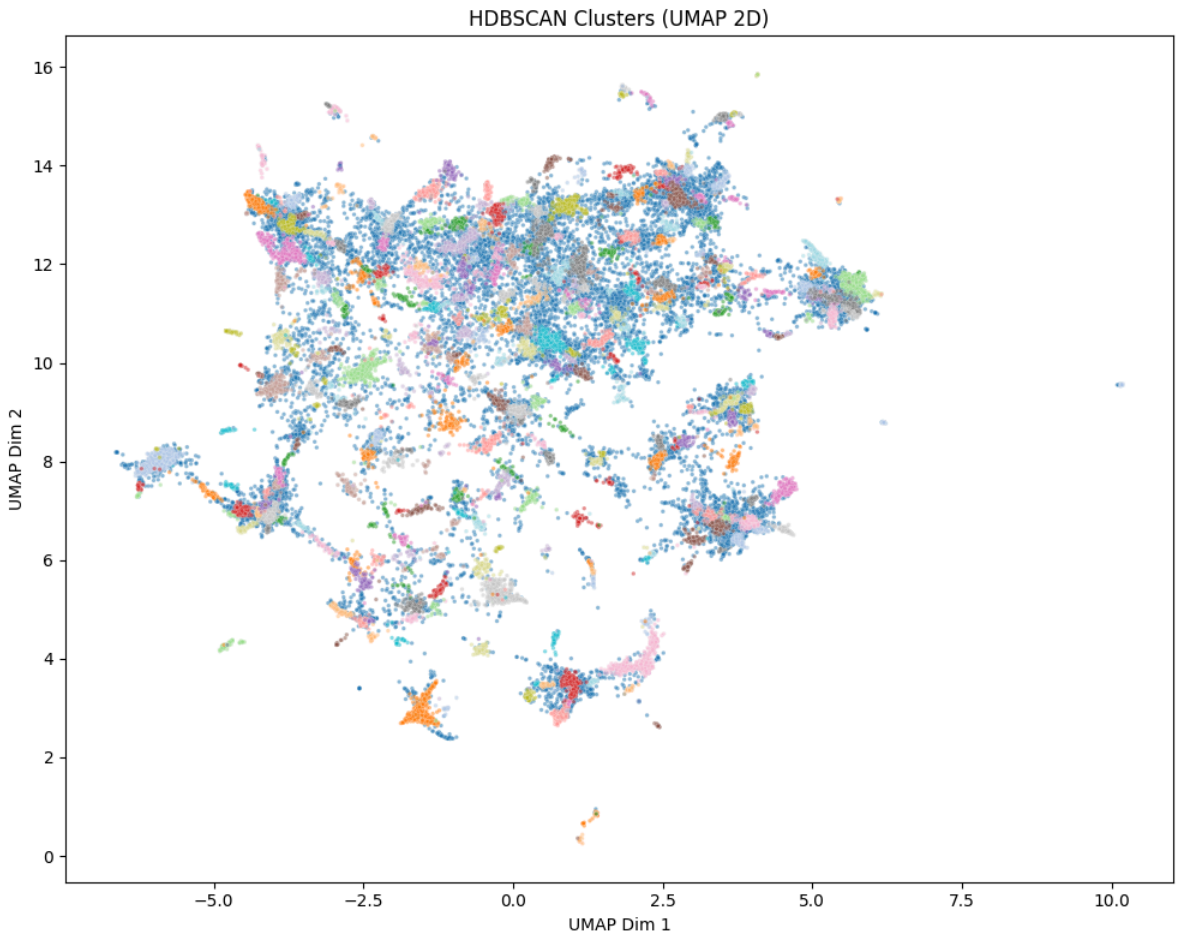
```
# Create DataFrame: x, y, cluster
tmp = pd.DataFrame(
    np.hstack([umap_2d, labels.reshape(-1,1)]),
    columns=["x", "y", "cluster"]
).sort_values("cluster")

# Convert cluster values to strings for coloring
tmp.cluster = tmp.cluster.astype(int).astype(str)

print("number of clusters plus NoiseCluster(-1) =", tmp["cluster"].nunique())
tmp.head()
```

```
number of clusters plus NoiseCluster(-1) = 351
```

	x	y	cluster
12030	3.281560	7.005876	-1
23066	-4.137316	14.109712	-1
23036	-0.872579	9.500083	-1
23037	-0.482952	13.124200	-1
23038	-3.319022	12.110134	-1

Next steps: Generate code with `tmp` New interactive sheet

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 8))

sns.scatterplot(
    data=tmp,
    x="x",
    y="y",
    hue="cluster",
    palette="tab20",
    s=6,
    alpha=0.5,
    legend=False    # ← no legend
)

plt.title("HDBSCAN Clusters (UMAP 2D)")
plt.xlabel("UMAP Dim 1")
plt.ylabel("UMAP Dim 2")

plt.tight_layout()
plt.show()


```



📘 Text Clustering Workflow — What We Accomplished

🔷 1. Load & Prepare the Dataset

- Pulled the ArXiv NLP dataset.
- Extracted **abstracts**, **years**, **categories**, and **titles**.
- These texts formed the foundation for the entire pipeline.

🤖 2. Convert Text → Embeddings

- Used a **SentenceTransformer (MiniLM)** to transform each abstract into a **semantic vector**.
- These embeddings capture the meaning of each document.

🔻 3. Reduce Dimensionality with UMAP

- Original embeddings were high-dimensional (hundreds of values).
- Applied **UMAP** to compress them into **5 dimensions**.
- This avoids the *curse of dimensionality* and improves clustering.

🧩 4. Cluster with HDBSCAN

- Used **HDBSCAN**, a density-based algorithm that:
  - Automatically finds cluster shapes
  - Detects outliers (noise)
  - Does *not* require you to choose the number of clusters
- Result: a set of meaningful cluster labels for each document.

🎨 5. Visualize Embeddings in 2D

- Used UMAP again (this time **n_components = 2**) for plotting.

- Plotted all documents in a scatter plot, colored by cluster.
- Allowed us to *see* how documents group together.

🔍 6. Inspect Cluster Meaning

- Sampled documents from selected clusters.
- Observed strong semantic themes—for example:
  - sarcasm detection
  - language modeling
  - NLP techniques
- Confirmed that HDBSCAN produced coherent groups.

🧠 7. Prepare for Topic Modeling

- After clustering, the next step (as in the textbook) is extracting **topic descriptions** from clusters.
- This leads to **BERTopic** or similar topic-modeling approaches.

⭐ End Result

You now have a full modern NLP clustering pipeline: **Embeddings → UMAP → HDBSCAN → 2D Visualization → Human Interpretation.**

Beautiful, modular, and completely aligned with the textbook.

Next step — build the BERTopic pipeline (clean & textbook-style)

Below is a short, simple implementation that follows the textbook pipeline:

1. embedding model → 2. UMAP for reduction → 3. HDBSCAN for clustering → 4. CountVectorizer → 5. c-TF-IDF → 6. (optional) KeyBERT-inspired representation.
   It uses GPU for embeddings when available and keeps everything minimal so you can paste it into your notebook.

```
pip install bertopic
```

```
Requirement already satisfied: numpy>=1.20.0 in /usr/local/lib/python3.12/dist-packages (from bertopic) (2.0.2)
Requirement already satisfied: pandas>=1.1.5 in /usr/local/lib/python3.12/dist-packages (from bertopic) (2.2.2)
Requirement already satisfied: plotly>=4.7.0 in /usr/local/lib/python3.12/dist-packages (from bertopic) (5.24.1)
Requirement already satisfied: scikit-learn>=1.0 in /usr/local/lib/python3.12/dist-packages (from bertopic) (1.6.1)
Requirement already satisfied: sentence-transformers>=0.4.1 in /usr/local/lib/python3.12/dist-packages (from bertopic) (5.1.2)
Requirement already satisfied: tqdm>=4.41.1 in /usr/local/lib/python3.12/dist-packages (from bertopic) (4.67.1)
Requirement already satisfied: llvmlite>0.36.0 in /usr/local/lib/python3.12/dist-packages (from bertopic) (0.43.0)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.12/dist-packages (from hdbscan>=0.8.29->bertopic) (1.16.3)
Requirement already satisfied: joblib>=1.0 in /usr/local/lib/python3.12/dist-packages (from hdbscan>=0.8.29->bertopic) (1.5.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.1.5->bertopic) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.1.5->bertopic) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=1.1.5->bertopic) (2025.2)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from plotly>=4.7.0->bertopic) (9.1.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from plotly>=4.7.0->bertopic) (25.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=1.0->bertopic) (3.6.0)
Requirement already satisfied: transformers<5.0.0,>=4.41.0 in /usr/local/lib/python3.12/dist-packages (from sentence-transformers>=0.4.1->bertc
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from sentence-transformers>=0.4.1->bertopic) (2.9.0+cu
Requirement already satisfied: huggingface-hub>=0.20.0 in /usr/local/lib/python3.12/dist-packages (from sentence-transformers>=0.4.1->bertopic)
Requirement already satisfied: Pillow in /usr/local/lib/python3.12/dist-packages (from sentence-transformers>=0.4.1->bertopic) (11.3.0)
Requirement already satisfied: typing_extensions>=4.5.0 in /usr/local/lib/python3.12/dist-packages (from sentence-transformers>=0.4.1->bertopic
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.12/dist-packages (from umap-learn>=0.5.0->bertopic) (0.60.0)
Requirement already satisfied: pynndescent>=0.5 in /usr/local/lib/python3.12/dist-packages (from umap-learn>=0.5.0->bertopic) (0.5.13)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->sentence-transformers>=0.4.1-
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->sentence-transformers
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->sentence-transformers>=0.4
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->sentence-transformers>=0.4.1-
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.20.0->sentence-transfor
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas>=1.1.5->bertopic) (1.17
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers>=0.4.1->bertopi
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers>=0.4.1->bert
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers>=0.4.1->be
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers>=0.4.1->bertopic)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transfc
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-trans
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transfc
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transforme
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transforme
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformer
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transform
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transfor
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transfor
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transform
```

```
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transfor
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transforme
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch>=1.11.0->sentence-transformers>=0.4.1->bert
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers<5.0.0,>=4.41.0->sentence-transfo
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers<5.0.0,>=4.41.0->senter
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers<5.0.0,>=4.41.0->sentence-transf
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch>=1.11.0->sentence-trans
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>=1.11.0->sentence-transformers>=0
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface-hub>=0.20.0->ser
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface-hub>=0.20.0->sentence-transf
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface-hub>=0.20.0->sentence-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface-hub>=0.20.0->sentence-
```

```python
# BERTopic pipeline (textbook-style, minimal)
import torch
from sentence_transformers import SentenceTransformer
from umap import UMAP
from hdbscan import HDBSCAN
from sklearn.feature_extraction.text import CountVectorizer
from bertopic import BERTopic
from bertopic.representation import KeyBERTInspired
from bertopic.vectorizers import ClassTfidfTransformer

# device-aware embedding model
device = "cuda" if torch.cuda.is_available() else "cpu"
embedding_model = SentenceTransformer("all-MiniLM-L6-v1", device=device)

```

```python

# dimensionality reduction (matches textbook)
umap_model = UMAP(n_neighbors=15, n_components=5, metric="cosine", min_dist=0.0)

# clustering
hdbscan_model = HDBSCAN(min_cluster_size=15, metric="euclidean", core_dist_n_jobs=4)

```

What This Part of the Pipeline Does

This section assembles the *topic representation* part of BERTopic — the components that turn raw clustered documents into human-readable topics.

---

1. CountVectorizer — Tokenizing & Counting Words

```python
vectorizer_model = CountVectorizer(stop_words="english", ngram_range=(1, 2))
```

- Converts each document into a bag-of-words matrix.
- Removes common English stopwords.
- Uses **1-grams and 2-grams**, so topics can include both single words and short phrases.

---

2. Class-TF-IDF — Scoring Words Inside Each Cluster

```python
ctfidf_model = ClassTfidfTransformer()
```

- Computes special TF-IDF scores **per cluster**, not per document.
- Highlights words that are *characteristic* for each topic.
- This is how BERTopic decides what the topic keywords are.

---

3. KeyBERTInspired — Improving the Topic Labels

```python
representation_model = KeyBERTInspired()
```

- Optionally refines the topic words using semantic similarity.
- Produces cleaner, more meaningful topic labels.

---

4. Assemble Everything into BERTopic

```python
topic_model = BERTopic(
    embedding_model=embedding_model,
    umap_model=umap_model,
    hdbscan_model=hdbscan_model,
    vectorizer_model=vectorizer_model,
    ctfidf_model=ctfidf_model,
    representation_model=representation_model,
    verbose=False
)
```

You plug the components together:

- **Embeddings → UMAP → HDBSCAN → CountVectorizer → c-TF-IDF → (Optional) KeyBERT**
- This forms the full customizable BERTopic pipeline.

---

5. Fit the Topic Model

```
topics, probs = topic_model.fit_transform(abstract_list)
```

- Embeds texts
- Reduces dimensions
- Clusters them
- Builds topic keywords
- Returns topic assignments + their probabilities

```

```

```
# tokenization / vectorization
vectorizer_model = CountVectorizer(stop_words="english", ngram_range=(1, 2))

# class-TF-IDF for topic scoring
ctfidf_model = ClassTfidfTransformer()

# optional representation model (gives nicer topic keywords)
representation_model = KeyBERTInspired()

# assemble BERTopic with the components
topic_model = BERTopic(
    embedding_model=embedding_model,
    umap_model=umap_model,
    hdbscan_model=hdbscan_model,
    vectorizer_model=vectorizer_model,
    ctfidf_model=ctfidf_model,
    representation_model=representation_model,
    verbose=False
)

```

```
# fit on your document list (e.g., 'abstract_list' from earlier)
topics, probs = topic_model.fit_transform(abstract_list)
```

probability = 0.0 for outlier In BERTopic Outliers always get prob=0.0 Because they aren't assigned to any stable topic. This is correct behavior.

```
for i in range(10):
  print(f"Document {i+1} belongs to topic {topics[i]} with probability {probs[i]}")
```

```
Document 1 belongs to topic -1 with probability 0.0
Document 2 belongs to topic -1 with probability 0.0
Document 3 belongs to topic -1 with probability 0.0
Document 4 belongs to topic -1 with probability 0.0
Document 5 belongs to topic 47 with probability 0.36878671497958343
Document 6 belongs to topic 272 with probability 1.0
Document 7 belongs to topic -1 with probability 0.0
Document 8 belongs to topic -1 with probability 0.0
Document 9 belongs to topic 54 with probability 0.4835514250072117
Document 10 belongs to topic -1 with probability 0.0
```

```
print("what each topic id corresponds to ?")
tmp2 = topic_model.get_topics()

for i in range(3):
    if i not in tmp2:
        print(f"topic {i+1}: (no such topic)")
        print("-" * 100)
        continue

    print(f"topic {i+1}: has the following words as its keywords:")

    # extract word + score
    g = [(t[0], float(t[1])) for t in tmp2[i]]

    for p in g:
        print(f"  ({p[0]}) with C-TFIDF {p[1]}")

```

```
    print("-" * 100)

```

```
what each topic id corresponds to ?
topic 1: has the following words as its keywords:
  (abstractive text) with C-TFIDF 0.430914044380188
  (extractive) with C-TFIDF 0.4030616283416748
  (summarization systems) with C-TFIDF 0.3898945748806
  (summarization models) with C-TFIDF 0.36866632103919983
  (abstractive summarization) with C-TFIDF 0.3609640598297119
  (document summarization) with C-TFIDF 0.36043453216552734
  (extractive summarization) with C-TFIDF 0.3522854447364807
  (abstractive summaries) with C-TFIDF 0.34827521443367004
  (summarization model) with C-TFIDF 0.34509730339050293
  (text summarization) with C-TFIDF 0.333077609539032
----------------------------------------------------------------------------------------------------
topic 2: has the following words as its keywords:
  (language processing) with C-TFIDF 0.45963090658187866
  (word embeddings) with C-TFIDF 0.45745375752449036
  (word embedding) with C-TFIDF 0.44197744131088257
  (bias nlp) with C-TFIDF 0.4281579852104187
  (language models) with C-TFIDF 0.42607101798057556
  (nlp) with C-TFIDF 0.331199586391449
  (quantify) with C-TFIDF 0.29845941066741943
  (languages) with C-TFIDF 0.2852994203567505
  (language) with C-TFIDF 0.2766522169113159
  (word) with C-TFIDF 0.2643653154373169
----------------------------------------------------------------------------------------------------
topic 3: has the following words as its keywords:
  (language processing) with C-TFIDF 0.6318051815032959
  (entity recognition) with C-TFIDF 0.5543075799942017
  (corpus) with C-TFIDF 0.4529123306274414
  (sequence labeling) with C-TFIDF 0.4273601174354553
  (annotation) with C-TFIDF 0.4214068055152893
  (languages) with C-TFIDF 0.3839644491672516
  (nlp) with C-TFIDF 0.3578883707523346
  (recognition ner) with C-TFIDF 0.3562939763069153
  (labeling) with C-TFIDF 0.3393876850605011
  (language) with C-TFIDF 0.3387593626976013
----------------------------------------------------------------------------------------------------
```

```
  1 tmp2
```

```
 ('words present', np.float32(0.32481116)),
 ('performance deep', np.float32(0.31187582)),
 ('representations answers', np.float32(0.30658963)),
 ('answer representations', np.float32(0.30543223)),
 ('representations', np.float32(0.29602608)),
 ('joint representation', np.float32(0.2926745)),
 ('encoders like', np.float32(0.2858916)),
 ('respondent representations', np.float32(0.28264552))]}
```

```
print("Representative documents for each topic:\n")

for topic_id in range(10):
    try:
        docs = topic_model.get_representative_docs(topic_id)
    except KeyError:
        print(f"Topic {topic_id}: (no such topic)")
        print("-" * 100)
        continue

    print(f"Topic {topic_id}:")
    print("Top representative documents:\n")

    for i, d in enumerate(docs):
        preview = d[:200].replace("\n", " ")  # shorten long abstracts
        print(f"  Doc {i+1}: {preview}...")

    print("-" * 100)

```

```
Representative documents for each topic:

Topic 0:
Top representative documents:

  Doc 1:   We propose a new length-controllable abstractive summarization model. Recent state-of-the-art abstractive summarization models basec
  Doc 2:   Despite significant progress, state-of-the-art abstractive summarization methods are still prone to hallucinate content inconsistent
  Doc 3:   Abstractive text summarization aims at compressing the information of a long source document into a rephrased, condensed summary. De
----------------------------------------------------------------------------------------------------
Topic 1:
Top representative documents:

  Doc 1:   Machine learning models are trained to find patterns in data. NLP models can inadvertently learn socially undesirable patterns when
  Doc 2:   Word embedding has become essential for natural language processing as it boosts empirical performances of various tasks. However, r
  Doc 3:   As Natural Language Processing (NLP) and Machine Learning (ML) tools rise in popularity, it becomes increasingly vital to recognize
----------------------------------------------------------------------------------------------------
Topic 2:
Top representative documents:

  Doc 1:   Most of the common applications of Named Entity Recognition (NER) is on English and other highly available languages. In this work,
  Doc 2:   Named entity recognition (NER) is an important research problem in natural language processing. There are three types of NER tasks,
  Doc 3:   Named Entity Recognition (NER) aims to extract and classify entity mentions in the text into pre-defined types (e.g., organization c
----------------------------------------------------------------------------------------------------
Topic 3:
Top representative documents:

  Doc 1:   The long-standing goal of Artificial Intelligence (AI) has been to create human-like conversational systems. Such systems should hav
  Doc 2:   To provide consistent emotional interaction with users, dialog systems should be capable to automatically select appropriate emotior
  Doc 3:   Emotion analysis has been attracting researchers' attention. Most previous works in the artificial intelligence field focus on recog
----------------------------------------------------------------------------------------------------
Topic 4:
Top representative documents:

  Doc 1:   In the wake of a polarizing election, the cyber world is laden with hate speech. Context accompanying a hate speech text is useful f
  Doc 2:   As research on hate speech becomes more and more relevant every day, most of it is still focused on hate speech detection. By attemp
  Doc 3:   The enormous amount of data being generated on the web and social media has increased the demand for detecting online hate speech. D
----------------------------------------------------------------------------------------------------
Topic 5:
Top representative documents:

  Doc 1:   Sentence-level relation extraction mainly aims to classify the relation between two entities in a sentence. The sentence-level relat
  Doc 2:   Joint entity and relation extraction framework constructs a unified model to perform entity recognition and relation extraction simu
  Doc 3:   Relation extraction aims to classify the relationships between two entities into pre-defined categories. While previous research has
----------------------------------------------------------------------------------------------------
Topic 6:
Top representative documents:

  Doc 1:   Attention-based methods and Connectionist Temporal Classification (CTC) network have been promising research directions for end-to-e
  Doc 2:   We present a state-of-the-art end-to-end Automatic Speech Recognition (ASR) model. We learn to listen and write characters with a jo
  Doc 3:   The acoustic-to-word model based on the connectionist temporal classification (CTC) criterion was shown as a natural end-to-end (E2E
----------------------------------------------------------------------------------------------------
Topic 7:
Top representative documents:

  Doc 1:   Prompt tuning has been an extremely effective tool to adapt a pre-trained model to downstream tasks. However, standard prompt-based
  Doc 2:   Prompt tuning is one of the successful approaches for parameter-efficient tuning of pre-trained language models. Despite being argua
  Doc 3:   Prompts for pre-trained language models (PLMs) have shown remarkable performance by bridging the gap between pre-training tasks and
```

Check out topic frequencies

```
1 topic_model.get_topic_freq()
```

	Topic	Count
0	-1	16603
28	0	700
86	1	628
39	2	571
43	3	562
...	...	...
253	351	15
335	352	15
276	353	15
88	354	15
158	355	15

357 rows × 2 columns

Check out C-TFIDF

```
 1 import pandas as pd
 2
 3 # 1. Extract sparse cTFIDF matrix
 4 ctfidf = topic_model.c_tf_idf_
 5
 6 # 2. Get vocabulary words
 7 vocab = topic_model.vectorizer_model.get_feature_names_out()
 8
 9 # 3. Convert to dense DataFrame
10 ctfidf_df = pd.DataFrame(ctfidf.toarray(), columns=vocab)
11
12 # 4. Add topic ID as a column
13 ctfidf_df['topic'] = ctfidf_df.index
14
15 ctfidf_df
```

	00	00 09	00 10	00 17	00 328	00 38	00 58	00 66	00 74	00 75	...	zyrian	zyrian furthermore	zyrian northern	zyrian skolt	zyx	zyx order	zyz0000	zyz fir
0	0.000028	0.0	0.0	0.000003	0.000003	0.000003	0.0	0.000003	0.0	0.0	...	0.000006	0.0	0.000003	0.000003	0.0	0.0	0.0	
1	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	
2	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	
3	0.000057	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	
4	0.000058	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
352	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	
353	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	
354	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	
355	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	
356	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.0	0.0	0.0	

357 rows × 1785185 columns

📘 Topic Modeling Workflow — Summary of What We Built

1. Generated Document Embeddings

We used a **SentenceTransformer** model (`all-MiniLM-L6-v2`) to convert each document into a dense vector. These embeddings capture semantic meaning → similar documents get similar vectors.

---

2. Reduced Embeddings with UMAP

UMAP compressed high-dimensional embeddings into a smaller space (e.g., 5D) while preserving structure. This step makes clustering easier, faster, and more stable.

---

3. Clustered Documents with HDBSCAN

HDBSCAN grouped reduced embeddings into topic clusters.

- Each cluster → one potential topic
- `-1` = outliers / documents that don't fit any cluster
- Produces soft probabilities (how well each document fits its cluster)

---

4. Vectorized Text with CountVectorizer

We transformed text into token counts (1–2 grams). This vocabulary is used for computing topic-word importance.

---

5. Built Topic Representations with c-TF-IDF

The **Class-TF-IDF** model computed how important each word is for each topic. This gives **topic-word weights**, not probabilities. Higher scores = more representative words.

---

6. (Optional) Refined Keywords with KeyBERTInspired

This step improved topic labeling by using embeddings to refine keyword quality.

---

7. Fit the Full BERTopic Pipeline

We assembled all components into a custom BERTopic model and ran:

```
topics, probs = topic_model.fit_transform(documents)
```

This produced: